MULTI-ROBOT PLATOONING IN HOSTILE ENVIRONMENTS
BY JEREMY SHIVELY

A Thesis
Presented to
The Academic Faculty

by

Jeremy Shively

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
May 2012

MULTI-ROBOT PLATOONING IN HOSTILE ENVIRONMENTS
BY JEREMY SHIVELY

Approved by:

Professor Magnus Egerstedt, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Ayanna Howard
School of Electrical and Computer Engineering
Georgia Institute of Technology

Professor Patricio Vela
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: May 2012

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The purpose of this thesis is to develop a testing environment for mobile robot experiments, to examine methods for multi-robot platooning through hostile environments, and test these algorithms on mobile robots. Such a system will allow us to rapidly address and test problems that arise concerning robot swarms and consequent interactions.

In order to create this hardware simulation environment a test bed will be created using ROS or Robot Operating System. This platform is highly modular and extensible for future development. Trajectory generation for the robots will use smoothing splines, B-splines, and A* search. Each method has distinct properties which will be analyzed and rated with respect to its effectiveness with regards to robotic platooning. A few issues to be considered include: Is the optimal path taken with respect to distance and threats? Is the formation of the robots maintained or compromised during traversal of the path? And finally, what sorts of compromises or additions are needed to make each method effective? This work will be helpful for choosing route planning methods in future work and will provide a large code base for rapid prototyping.

CHAPTER I

INTRODUCTION

1.1 Motivation

From toys to tanks, the field of robotics has become highly integrated into our modern culture. Currently a more advanced task requires human guidance to complete, but the natural progression for machines is to become more automated with time. The state of robotics is beginning to have a significant presence in today's military. As the capabilities of the robotics field increases, and the implementation costs decrease, we will see a greater number of human-centered activities being replaced by autonomous robots.

This is beneficial in a number of ways. Current trends reveal the increasing frequency of robots assuming tasks normally reserved for humans. As technology improves, certain mundane, repetitious or even dangerous jobs have been passed off. An example of such a situation has become the starting point for this research project. In potentially dangerous environments, a robotic convoy could be sent out to assist or even replace a human fire squad, thereby reducing the risk of injury or death.

In this hypothetical situation of a robotic convoy, multiple robots would be assigned to travel together in a line or other formation with the goal of going from Point A to Point B, avoiding any threats along the way[1][2]. Such a situation is the motivation for the following research.

1.2 Problem Statement

In order to study the interactions between a robotic convoy, the environment, and individual robots, a hardware based simulation environment is needed. Many topics including navigation, group coordination, and communication would benefit from additional researched.

In these types of studies it is useful to start with a quick software simulation then test these ideas with a hardware based simulation. Between software simulation and actual implementation, a hardware simulation is useful for identifying flaws such as not accounting for limitations of hardware. It is also useful for developing code that can, with very few changes, be applied to the final robots in the actual environments. With a hardware simulation environment setup, the first issue to address is navigation of robots. Various navigation techniques need to be implemented and evaluated not only for characterization but to be available for future projects. Some of the environments will be empty and some will contain hostile threats. A hostile environment will be defined as an environment containing areas which if entered may compromise the mission.

1.3 Proposed Work

The two main objectives of this thesis are to develop a hardware simulation environment and characterize navigation methods. The hardware environment will consist of a central computer running the software and calculating the trajectories, a motion tracking system for location data, and mobile robots simulating vehicles with nonholonomic properties. Further details of the environment will be discussed in Chapter 2. The navigation methods will be created for multi-robot platooning, with particular emphasis on path planning through hostile environments. These methods will then be tested on the mobile robots. Multiple path planning algorithms to be tested and examined in this environment, with the purpose of showing what changes need to be made in order to complete the assigned tasks for the various algorithms. All of this will be done is such a way to allow rapid development for future work.

The first technique that will be attempted are splines. Two types of splines will be examined, smoothing splines and B-splines. Smoothing splines are an optimal control problem which tries to balance how close the trajectory comes to the waypoints with the control input needed to get there. This tunable parameter which balances these two factors

will be referred to as either Rho, or the smoothing parameter. As the name implies this rho value controls the smoothness of the line simply because a smooth path requires less energy to follow than a path with excessive disturbances. Such a path can be useful when the main consideration is energy expenditure, or when smaller path disturbances need to be averaged into a smoother path. The second type of spline that will be used is the cubic B-spline. B-Splines are essentially calculated by a few matrix multiplications making them very quick to calculate. They can also be made continuously differentiable in the third degree by deriving the B-Spline equations with third degree polynomials. The advantage of this is the acceleration and velocity components of the vehicle will be continuous and not make any sudden jumps that the hardware is not capable of. The trajectory of both splines will need to be fine-tuned for threat avoidance by adjusting points that go through threat zones. It will then be examined how well this works for simple and complex environments.

The next method for trajectory generation that will be implemented is a graph search, specifically A*. The theory of the A* optimal path algorithm has been well studied but the algorithm has been designed to work for robots capable of moving in any direction or at least run at a resolution where nonholonomic constraints were not taken into consideration. This method will be compared to generating paths using splines with comparisons between computation time and the route taken. Questions that will be studied are:

- Is the optimal path taken?
- Does the optimal path cause the robots to leave their formation?
- Does this path put the robots in an environment of high risk based on surrounding terrain or threats?

Once paths have been generated for each method they will be overlaid with each other to easily see the differences of the resulting paths. Performance metrics will include total distance and calculation time, and be used to analyze the differences between the results.

These measurements will help determine the advantages and disadvantages of each method. The location of threats will be provided to the convoy as that is the flow of information used in the parent project. The difficulties of this thesis will be within the implementation of the algorithms and creating an environment that adequately tests each method.

CHAPTER II

HARDWARE AND SOFTWARE USED

In order to study situations involving autonomous vehicles a system needs to be created which can as accurately as possible mimic all the properties of the full scale situation. While not all characteristics of the full scale system can be accurately incorporated into the smaller simulation, many choices can be made to decrease the difference between the two. The first step towards decreasing these differences is selecting hardware which can be setup in such a way as to recreate characteristics from communication networks, navigational limitations, and sensing capabilities. In this chapter each major component will be described along with the reasons why it was used and how it helps produce results which are easily generalizable.

2.1 KheperaIII (UGV)

The KheperaIII, shown in Figure 1, is a small robot platform which runs an embedded Linux operating system. It contains eight infrared proximity sensors, two infrared ground sensors, and five ultrasonic sensors. Its small size and plethora of sensors make it ideal for swarm robot experiments. A full size autonomous vehicle would also be surrounded by sensors to learn about the local environment, so the sensors on our system can recreate this characteristic. Additionally a wireless compact flash card has been added to each of the Khepera robots. With each robot capable of communication various types of networks can be created ranging from one robot being the leader to every robot knowing everything. With this combination of hardware it is possible to either run necessary software on the robots or run a driver on the robot that listens for commands from a central computer. The latter method is the technique we will be using for this project due to the flexibility it allows when making changes. Even though the code is running on a single computer,

each KheperaIII has its own node which acts as the software equivalent to running on the Khepera itself. This provides a tremendous advantage because of the ability to analyze communication between nodes.

Figure 1: The KheperaIII

2.2 Vicon

Location data is important for the calculation of relative displacement to other vehicles and threats. This data is obtained using the Vicon Motion Capture System and acts as an indoor GPS system. The main components of this system are the cameras, data receiver, and Vicon iQ software. For this lab we have eight cameras, four are placed in each corner of the room and four are placed along each wall. The data acquired by the cameras is sent to the data receiver where is converted to a format that is easily read by the computer software. The computer then broadcasts UDP packets on the network containing X, Y and Z position data along with angle data relative to the X, Y and Z axes. While the sensors on the Kheperas can provide information about obstacles to avoid, the Vicon system was selected to fill this roll. The reason for this is to provide a single interface for location data and obstacle data. The Vicon system provides locations comparable to what GPS would provide full scale autonomous systems.

2.2.1 Vicon Limitations

Even after sufficient calibration the accuracy of the system is sometimes an issue. Limitations of the Vicon system during testing appear to be accuracy, and coverage area. There is

sometimes a disagreement on the position of a reflective reference point on a robot, and as a result, the positive Z axis is oriented in the negative Z direction. The second limitation of the Vicon system is the coverage area provided by the setup. The room that the system is located in is 9 meters by 7.6 meters. With the Vicon cameras situated on tripods at the edge of the room, the coverage area of the system is 5 meters long by 5 meters by 1.6 meters high. This creates a severe limitation when testing multiple robots, especially when several flying robots are involved. In this experiment, the height limitation will not be an issue, however the relatively small coverage area means the robots cannot travel any great length in one direction. Because of this area restriction the complexity of generated paths is also limited especially when following these paths with multiple robots in formation.

2.3 ROS (Robot Operating System)

ROS stands for Robot Operating System and is an open source project currently in active development. Many of the core features and API's are developed by Willow Garage, while code contributions are constantly being made by the community. The advantage of ROS over developing a system from the ground up is the roscore which provides a convenient and reliable way for processes to communicate. The roscore also allows for measurements such as bandwidth being used between nodes thus allowing for fine grain study of the system. These and other advantages will be described in further detail below. The version used for this project is ROS Diamondback.

2.3.1 Packages

Packages are the main grouping method for software developed for ROS. A package can be developed and run on any computer with ROS installed with little or no changes. Each package contains one or more related nodes that work together to provide certain functionality. The package system keeps the code modular and able to be used on different systems. All software used in this project is contained within two packages, the Vicon Package and the KheperaIII Package. The package can be abstracted out to represent different features

on the full scale system. Such modularity is important because in a convoy different types of vehicles may be used and each type of vehicle can be represented by a different package. Each type of autonomous vehicle can have its own collection of packages which represent the various capabilities of that vehicle.

2.3.2 Nodes

Within each package there can be many nodes; each node being equivalent to a single process in a program. Since all nodes communicate with stateless connections, it is not required for all nodes to run at once. In other words, each node is able to be run independently of all other nodes, with the exception of the roscore. This allows for easy isolation when debugging individual nodes. The communication between nodes is also able to be monitored for additional help during debugging or to ensure the network communication is within specified limitations.

Examples of tasks that are implemented as nodes include each of the path generation methods. This allowed us to switch generation methods while continuing to run all other code. Another example is the controller for each robot. Selecting the controller used is just a matter of starting another node with different parameters. It is this modularity which creates an easy to understand code base and makes for little start-up overhead for future development.

2.3.3 Communication

In ROS, nodes communicate via services or messages. Services are a request/reply type of communication. Information is sent as a request, which the node processes and returns the results in the reply. This type of interface is used for two way communication between nodes. However, some instances only require communication to go one-way. In such cases, messaging would be used. With messaging, nodes have the ability to publish and subscribe to a topic. When a node publishes a message, all nodes that are subscribed to this topic are notified and thus are able to immediately make use of the new data. All

running services and message topics are tracked by the parameter server within the roscore. ROS also provides many different tools for examining communications between nodes such as data rates, lists of available topics and services, and graphical representations of the communication network.

2.4 Layout Overview

The motion tracking software will be run on an independent desktop that is connected to each camera over Ethernet. The software analyzes the images and triangulates the coordinates of each mobile robot. These coordinates are then broadcast on the wireless network using UDP.

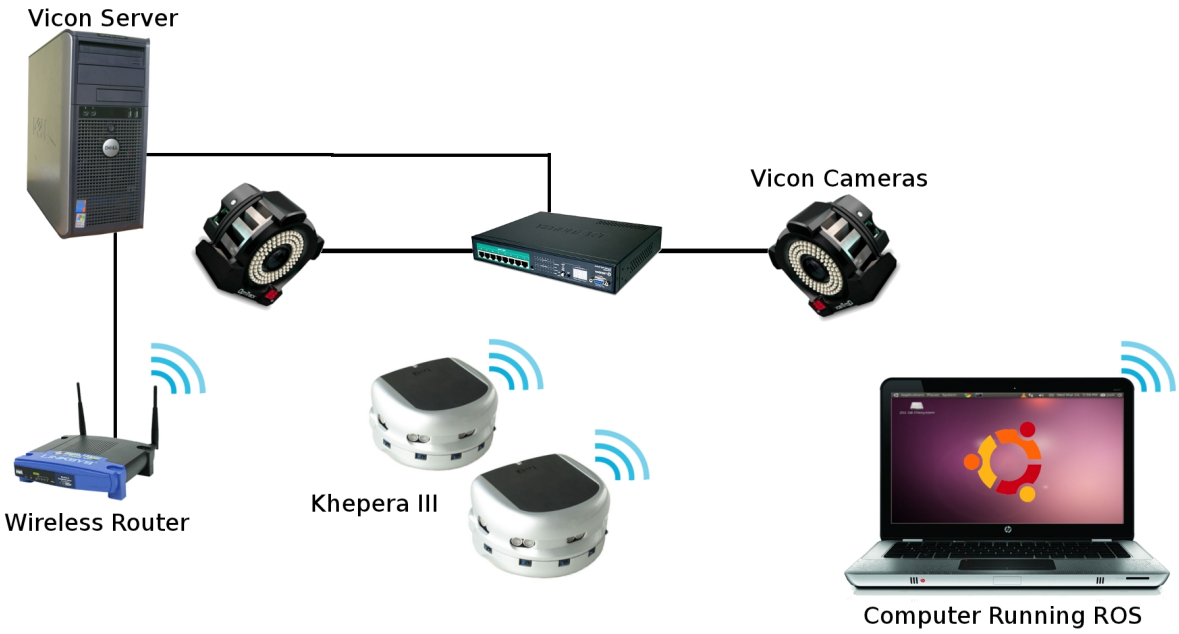

Figure 2: Hardware Layout

The motion capture node takes the UDP packets that are broadcast on the network and publishes them to a topic so the locations are available for the other nodes to use. The two nodes that use these coordinates are the program core and the controller. These two nodes are run in their own namespace for each Khepera that is being controlled. This makes the current architecture extensible to being run on each Khepera robot individually. If the

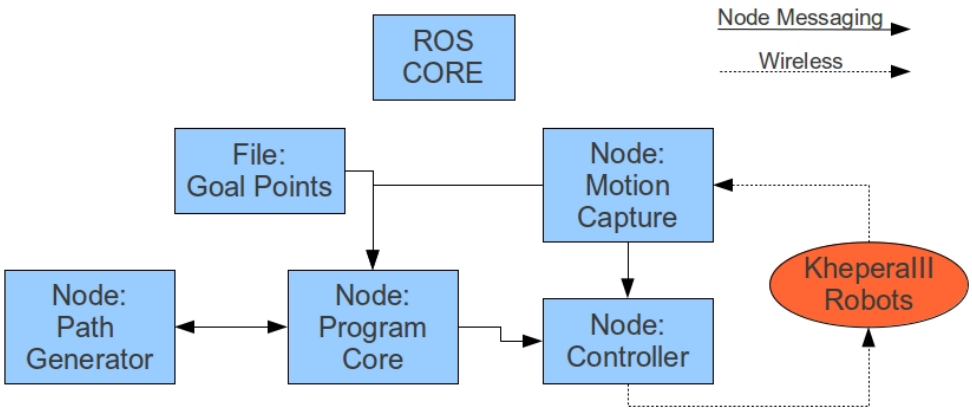

Figure 3: The Program Architecture

program core is assigned to a follower it will calculate the needed destination coordinates based on the leader position, and then send this destination to the controller. Alternatively if the program core is assigned to the Khepera leader, it will request a trajectory from the path generator node, and forward the next destination point in that path to the controller. The controller will then take this destination point and current robot location and find the necessary control inputs to get the Khepera to its destination. Once the robot is within a given distance of the destination point the program core will send the coordinates of the next destination point. This cycle continues with the program core requesting new trajectories as needed.

There are multiple reasons why this configuration of hardware and software make for a good representative system. First, because of the ample sensors on the Khepera robots they are able to understand their local environment and react accordingly similar to real systems. Second, because of the modular ROS system, fine tuning and testing is capable to ensure network and processing limitations are respected. Third network and environmental characteristics are able to be adequately represented. Together these properties allow for accurate recreation of full scale autonomous systems while minimizing the differences between the two.

CHAPTER III

FORMATION CONTROL

3.1 Controls

The controllers for the Khepera robots are designed to accept X and Y coordinates as inputs and return the change in speed and heading. The controller used is a gradient descent controller which allows the robot to loosely follow any trajectory that does not fit within the dynamics of the system. Since vehicles have a smaller turning radius at slower speeds, this property was incorporated into the controllers. If a destination point is located directly in front of the Khepera the speed will be at its highest value. Alternately, if a destination point is located 90° to the heading of the vehicle, the forward speed will be at a minimum but greater than zero to preserve the characteristics of an autonomous car.

3.1.1 Leader Control

The controller for the leader accepts the coordinates provided by the trajectory generation and treats them as the local minimum to head towards. The controller sends the change in speed and change in heading to the robot to reach this local minimum. Once this point is within an acceptable radius the next point in the trajectory is treated as the minimum. The speed gains are tuned to be slightly lower than follower speed gains to guarantee followers can maintain their distance from the leader.

3.1.2 Follower Control

The main difference between the follower controller and the leader controller is the source of the destination. Instead of pulling this destination from the trajectory generator like the leader, the follower controller simply uses the location of the robot it is following. The second change to the follower controller is instead of trying to reach the destination

provided it tries to reach a specified radius away from the provided destination. This results in a predetermined barrier to be maintained between the robots.

3.2 Control Equations

The system dynamics of a unicycle like robot are given below. V is the velocity scalar and θ is the angle of the robot.

$$\dot{x} = V\cos\theta \tag{1}$$

$$\dot{y} = V\sin\theta \tag{2}$$

$$\dot{\theta} = \omega \tag{3}$$

$\dot{x}$ and $\dot{y}$ can be represented as a vector by combining $cos(\theta)$ and $sin(\theta)$ into vector $h(\theta)$

$$h(\theta) = \begin{bmatrix} cos(\theta) \\ sin(\theta) \end{bmatrix} \tag{4}$$

This allows the system dynamics to be represented as follows.

$$\dot{p} = Vh(\theta) \tag{5}$$

$$\dot{\theta} = \omega \tag{6}$$

One controller to try for the leader is the following where J is the rotation matrix and r is the distance to the goal.

$$\dot{\omega} = -k_1 r h' J u \tag{7}$$

$$\dot{V} = k_2 h' u \tag{8}$$

The angular velocity is proportional to distance from the goal and angle from the goal. The linear velocity is proportional to distance from the goal but inversely proportional to angle

from the goal.

$$r = \sqrt{(x - g_x)^2 + (y - g_y)^2} \tag{9}$$

With the matrices and vectors of the $\dot{\omega}$ filled in, the terms can be reduced to a single equation of scalars to be used in the code, as seen in equation 10.

$$\dot{\omega} = -k_1 r \begin{bmatrix} h_x & h_y \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \left(\begin{bmatrix} g_x \\ g_y \end{bmatrix} - \begin{bmatrix} x \\ y \end{bmatrix} \right)$$

$$\dot{\omega} = -k_1 r \begin{bmatrix} -h_y & h_x \end{bmatrix} \begin{bmatrix} g_x - x \\ g_y - y \end{bmatrix}$$

$$\boxed{\dot{\omega} = -k_1 r(-h_y g_x + h_y x + h_x g_y - h_x y)} \tag{10}$$

The main difference in equation 11 is the absence of the parameter based on the distance to the goal. This means the leader's speed will not be affected by how far away the goal is. The other difference is the absence of the rotation matrix. This is because the magnitude of these two values are inversely related. When the robot is turning quickly it is traveling forward slowly however when it is traveling forward quickly, it has little rotation.

$$\dot{V} = k_2 h' u$$

$$\dot{V} = k_2 \begin{bmatrix} h_x & h_y \end{bmatrix} \begin{bmatrix} g_x - x \\ g_y - y \end{bmatrix}$$

$$\boxed{\dot{V} = -k_2(-h_x g_x - h_x x + h_y g_y - h_y y)} \tag{11}$$

The follower equations are the same as the leader except they have been slightly modified to maintain a specified distance away from the goal. Since the goal being sent to the followers is the position of the leader, the followers maintain a given distance away from the leader. The follower equations have been calculated as follows.

$$f = \text{Distance to maintain}$$

$$\nabla V x = 2(r - f)(x - g_x)$$

$$\nabla V y = 2(r - f)(y - g_y)$$

The equations for ∇V shown above are used to simplify equation 13. The angular velocity equation remains the same as the leader equation except for now taking into account the distance to maintain as can be seen in equation 12.

$$\dot{\omega} = -k_1(r - f)h' J u \tag{12}$$

$$\dot{\omega} = -k_1(r - f)\begin{bmatrix} -h_y & h_x \end{bmatrix}\begin{bmatrix} g_x - x \\ g_y - y \end{bmatrix}$$

$$\dot{\omega} = -k_1(h_x \nabla V_y - h_y \nabla V_x) \tag{13}$$

In the follower speed value the distance to the goal subtracted by the distance to maintain is included for two reasons. If the follower is farther away, it is able to catch up quickly and once the distance to maintain has been reached the change in speed goes to 0 so the follower maintains the speed of the leader.

$$\dot{V} = -k_2(r - f)h' u \tag{14}$$

$$\dot{V} = -k_2(r - f)\begin{bmatrix} h_x & h_y \end{bmatrix}\begin{bmatrix} g_x - x \\ g_y - y \end{bmatrix}$$

$$\dot{V} = -k_2(h_x \nabla V_x + h_y \nabla V_y) \tag{15}$$

The angular velocity of the follower should be based on the orientation relative to the leader (h), the distance away from the goal (u), and a gain based on how close the follower is from the leader ($r - f$). The linear velocity is based on the same properties, however, it does not include the rotation matrix like the angular velocity. The reason for this is the desired speed based on the orientation. For example, for a given distance if the robot is facing the leader we want a large linear velocity and a small angular velocity. The inverse is also true; if the robot is facing away from the leader we want a small linear velocity and a large angular velocity.

With these controllers implemented it is possible to run the system using the waypoints as the inputs to the controllers. The disadvantage to this is only one waypoint at a time is considered. This means the heading of the robot is not balanced between the current segment and the next segment but rather is trying to finish the current segment as fast as possible. This is why path planning is needed in addition to the gradient descent controller. Path planning algorithms will not only allow the robots to take into consideration future waypoints, when planning the current path, but will also allow them to take obstacles into consideration. In chapter 4 a few different types of planning methods will be added to the current system in order to increase performance and efficiency.

CHAPTER IV

TRAJECTORY GENERATION/REPLANNING

4.1 Path planning overview

The purpose of path planning is to provide a route between a starting and finishing point while avoiding any obstacles along the way. There are many different algorithms that can be used such as Graph Searching[3][4], Splines[5], Sampling-Based algorithms[6][7], or Combinatorial Planning[8].

Only a few points the operator would like the vehicle to go to and a series of points will be generated such that a path is generated which intersects or passes close to each specified waypoint. The second reason for using trajectory generation is because the controller just provides a simple potential field for the vehicle to follow and runs the risk of not reaching its destination in an environment with obstacles and local potential minimums. This is where various trajectory methods are needed. Smoothing Splines will be used for their ability to balance the importance of intersecting the waypoint with the energy needed to get there. B-Splines will be used to maintain a continuous function at a high enough degree for the robot to correctly follow. Astar will be used because of its ability to find the optimal path even when obstacles are present. All these methods will be described in further detail below.

4.1.1 Smoothing Splines

Smoothing Splines are a subset within Splines which do not guarantee waypoint intersection but optimize the path for a specified cost[9]. In the following examples we have optimized the path in terms of input energy needed to follow the path. The X and Y terms weqre kept separate to maintain a higher level of controllability in the system. However,

describing the system with a set of parametric equations does not provide complete controllability of the generated path. As a result it is not possible to control the radius of curves created and thus cannot guarantee that the path generated will be possible for a nonholonomic robot to follow. This problem becomes apparent when waypoints are close together or the next waypoint is behind the current heading.

4.1.2 B-Splines

Unlike smoothing splines, B-Splines are a piecewise-polynomial function. The main difference between these two methods is waypoint intersection. B-Splines are guaranteed to pass through the waypoints, while smoothing splines only try to come close. The B-Spline examples in this project are also parametric, like the smoothing splines, as opposed to having Y in terms of X. The B-Spline equations were derived using third order polynomials. This means that the trajectories are continuous to the third degree which allows for the robot hardware to accurately follow the path assuming it is within the turning radius bounds of the vehicle.

4.1.3 A* Searching

A* search is a very different method of path generation from Splines. The main advantage of A* is that it will find the optimal path to a goal. However this path may not take the vehicle dynamics into consideration and could result in a path that is not physically possible to follow. A* is a very generic search algorithm and many advancements have been made to speed this algorithm up. In this thesis the A* method will be used as a base comparison for shortest distance.

4.1.4 Comparison

By their very nature splines are continuously differentiable to a given degree because they are the melding of two or more polynomials of the specified degree. Smoothing splines and B-spline generate nonholonomic paths however the rate of change for turning is not able to

be parameterized and thus not able to be constrained to a vehicle with specific properties. A* is often implemented at a resolution which abstracts out the problem of turning radius and vehicle boundary conditions.

Another major issue is obstacle avoidance. This is a feature which is not incorporated into standard spline generation however various methods have been used to shift and re-shape splines such that they avoid obstacle. Though after reshaping the path the properties of the spline may no longer hold true and tests for continuously differentiable to a given degree may need to be implemented. A* is designed with obstacles in mind and upon reaching an obstacle simply marks that area as impassable and continues to search for the shortest route. The reason we can use A* as a base comparison is because it always uses the discretized optimal path and we can calculate a lower bound on distance traveled.

Table 1: Properties

Properties	**Smoothing Spline**	**B-Spline**	**A* Search**
Continuously differentiable	✓	✓	
Fits unicycle constraints by default			
Fits unicycle constraints with adjustments			✓
Avoids obstacles by default			✓
Avoids obstacles with adjustment	✓	✓	✓
Average computational complexity	O(n)	O(n)	O(nlog(k))

The computational complexity of both splines increases linearly with the number of waypoints while the complexity of Astar increases at best by nlog(k) with n being the number or waypoints and k being the discretization parameter.

4.2 Derivations

4.2.1 Smoothing Splines

Start by assuming a linear, time-invariant system.

$$\begin{aligned} \dot{x}(t) =& Ax(t) + bu(t) \\ y(t) =& cx(t) \end{aligned} \tag{16}$$

Solving these differential equations results in the following equation.

$$y(t) = ce^{At}x_o + \int_0^T ce^{A(t-s)}bu(s)\,\mathrm{d}s \tag{17}$$

Find Gramian

$$\ell_t(s) = \begin{cases} ce^{A(t-s)}b & t > s \\ 0 & \text{Otherwise} \end{cases}$$

$$G = \int_0^T \ell(s)\ell^T(s)\,\mathrm{d}s \tag{18}$$

The next step is to find the optimal control coefficients. The following linear equation must be solved for τ in order for the system to have an optimal control signal. W is a diagonal matrix of positive weights that can be used to determine the relative importance of each segment in reaching the waypoint. G is the gramian calculated above and ρ is the smoothing parameter vector.

$$(WG + \rho I)\tau = W\alpha \tag{19}$$

$$\tau = [(WG + \rho I)]^{-1}W\alpha \tag{20}$$

With the τ vector now solved, the optimal control signal can now be calculated.

$$u(t) = \tau^T \ell_T(t) \tag{21}$$

The smoothing spline can now be calculated by substituting the optimal control signal back in the system dynamics.

$$\dot{X} = Ax + bu \tag{22}$$

4.2.2 B-Splines

The formation of the B-Spine is from the joining of multiple polynomials. In this situation this is done by matching the coordinates and first and second derivatives at the union of each polynomial. What this means is that each polynomial can be represented by three equations and these equations have to be equal at that one point. The derivation of B-Splines is as follows starting with the equations for polynomials.

Parametric polynomial equations

$$x = a_x(t)^3 + b_x(t)^2 + c_x(t) + d_x \qquad (23) \qquad y = a_y(t)^3 + b_y(t)^2 + c_y(t) + d_y \qquad (24)$$

First derivative of the polynomial equations

$$\dot{x} = 3a_x t^2 + 2b_x t + c_x \qquad (25) \qquad \dot{y} = 3a_y t^2 + 2b_y t + c_y \qquad (26)$$

Second derivative of the polynomial equations

$$\ddot{x} = 6a_x t + 2b_x \qquad (27) \qquad \ddot{y} = 6a_y t + 2b_y \qquad (28)$$

Where t represents the time between knots which are the points in time where the spline will merge with each other. The following equations are for the second order boundary conditions of the first polynomial, where s_1 is the first knot and s_2 is the second knot.

$$\ddot{x}_1 = 6a_{x1}(s_1 - s_1) + 2b_{x1} = 2b_{x1} \qquad \ddot{y}_1 = 6a_{y1}(s_1 - s_1) + 2b_{y1} = 2b_{y1}$$

$$\ddot{x}_2 = 6a_{x1}(s_2 - s_1) + 2b_{x1} \qquad \ddot{y}_2 = 6a_{y1}(s_2 - s_1) + 2b_{y1}$$

From these two sets of equations we solve for a_1 and b_1 in terms of $\ddot{x}$ and $\ddot{y}$ with c_1 and d_1 still unknown. The equations will also be simplified by representing $s_2 - s_1$ with h_1

$$a_{x1} = \frac{\ddot{x}_2 - \ddot{x}_1}{6h_1} \qquad (29) \qquad a_{y1} = \frac{\ddot{y}_2 - \ddot{y}_1}{6h_1} \qquad (30)$$

$$b_{x1} = \frac{\ddot{x}_1}{2} \qquad (31) \qquad b_{y1} = \frac{\ddot{y}_1}{2} \qquad (32)$$

With equations 29 through 32 we substitute these equations into 27 and 28 to get the following second derivative equation for the first union point.

$$\ddot{x} = \frac{(x - x_1)(\ddot{x}_2 - \ddot{x}_1)}{x_2 - x_1} + \ddot{x}_1 \qquad\qquad \ddot{y} = \frac{(y - y_1)(\ddot{y}_2 - \ddot{y}_1)}{y_2 - y_1} + \ddot{y}_1$$

Now we constrain our polynomials to pass through the waypoints with previously unused equations 23 and 24. We do this at the first and second knot, and set the equations for each knot equal to each other to solve for the coefficients.

$$c_{x1} = \frac{x_2 - x_1}{h_1} - \frac{\ddot{x}_2 h_1}{6} - \frac{\ddot{x}_1 h_1}{3} \qquad (33) \qquad\qquad c_{y1} = \frac{y_2 - y_1}{h_1} - \frac{\ddot{y}_2 h_1}{6} - \frac{\ddot{y}_1 h_1}{3} \qquad (34)$$

$$d_{x1} = x_1 \qquad (35) \qquad\qquad d_{y1} = y_1 \qquad (36)$$

At this point in the derivation all coefficients have been solved in terms of x, y, $\ddot{x}$ or $\ddot{y}$. The next step is to solve for $\ddot{x}$ and $\ddot{y}$ which is done by constraining the boundary conditions using the two first derivative equations 25 and 26.

$$h_1\ddot{y}_1 + 2(h_1 + h_2)\ddot{y}_2 + h_2\ddot{y}_3 = 6\left(\frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1}\right) \qquad (37)$$

$$h_1\ddot{x}_1 + 2(h_1 + h_2)\ddot{x}_2 + h_2\ddot{x}_3 = 6\left(\frac{x_3 - x_2}{h_2} - \frac{x_2 - x_1}{h_1}\right) \qquad (38)$$

which written in a general matrix form representing all the segments is

$$H\begin{bmatrix} \ddot{x}_1 & \ddot{y}_1 \\ \ddot{x}_2 & \ddot{y}_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} = 6\begin{bmatrix} 0 & 1 \\ \frac{x_3 - x_2}{h_2} - \frac{x_2 - x_1}{h_1} & \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \vdots & \vdots \\ \frac{x_n - x_{n-1}}{h_{n-1}} - \frac{x_{n-1} - x_{n-2}}{h_{n-2}} & \frac{y_n - y_{n-1}}{h_{n-1}} - \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \\ 0 & 1 \end{bmatrix} \qquad (39)$$

$$H = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ h_1 & 2(h_1 + h_2) & h_2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \dots & 0 & 0 & 1 \end{bmatrix} \tag{40}$$

Each knot is given at a specified time ranging from 0 to 1. The difference between each knot will be designated by a value of h. For example $h_1 = s_2 - s_1$. The knots can be spaced equally across 0 to 1 or they can be distributed in such a way to allow for greater flexibility between certain points. With these values set we create our H matrix and multiply both sides by H^{-1}.

$$\begin{bmatrix} \ddot{x}_1 & \ddot{y}_1 \\ \ddot{x}_2 & \ddot{y}_2 \\ \vdots & \vdots \\ x_n & y_n \end{bmatrix} = 6H^{-1} \begin{bmatrix} 0 \\ \frac{y_3 - y_2}{h_2} - \frac{y_2 - y_1}{h_1} \\ \vdots \\ \frac{y_n - y_{n-1}}{h_{n-1}} - \frac{y_{n-1} - y_{n-2}}{h_{n-2}} \\ 0 \end{bmatrix} \tag{41}$$

This generic equation gives the needed values for $\ddot{x}$ and $\ddot{y}$ from any set of waypoints. With the coefficients being in terms of x, y, $\ddot{x}$ and $\ddot{y}$, all of which are now known. With these values all coefficients from equations 23 and 24 can now be calculated using the generic versions of equations 29 through 36.

$$a_{xn} = \frac{\ddot{x}_{n+1} - \ddot{x}_n}{6h_n} \qquad a_{yn} = \frac{\ddot{y}_{n+1} - \ddot{y}_n}{6h_n}$$

$$b_{xn} = \frac{\ddot{x}_n}{2} \qquad b_{yn} = \frac{\ddot{y}_n}{2}$$

$$c_{xn} = \frac{x_{n+1} - x_n}{h_n} - \frac{\ddot{x}_{n+1}h_n}{6} - \frac{\ddot{x}_n h_n}{3} \qquad c_{yn} = \frac{y_{n+1} - y_n}{h_n} - \frac{\ddot{y}_{n+1}h_n}{6} - \frac{\ddot{y}_n h_n}{3}$$

$$d_{xn} = x_n \qquad d_{yn} = y_n$$

With the knot values and the coefficients known we can calculate the polynomials and piece together the desired spline by multiplying the matricies shown in equation 42.

$$\begin{bmatrix} T_1^3 & T_1^2 & T_1 & 1 \\ T_2^3 & T_2^2 & T_2 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ T_{n-1}^3 & T_{n-1}^2 & T_{n-1} & 1 \\ T_n^3 & T_n^2 & T_n & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_{n-1} & y_{n-1} \\ x_n & y_n \end{bmatrix} \tag{42}$$

All times that are used to calculate the spline must fall within the range of the knot values. The two knot values that the time falls between are what is used to determine which set of coefficients are used. For example if T falls between k_1 and k_2 then a_1, b_1, c_1 and d_1 are used. This means that once the coefficients have been calculated the granularity of the spline points can be easily adjusted.

4.2.2.1 Controlling the Turn Radius

To control the turn radius we must be able to control the second derivative of the spline $\frac{d^2y}{dx^2}$

$$\frac{d^2y}{dx^2} = \frac{d(dy/dx)}{dx} \tag{43}$$

$$= \frac{d(\dot{y} * 1/\dot{x})}{dx} \tag{44}$$

$$= \frac{1}{\dot{x}}\frac{d(\dot{y})}{dx} + \dot{y}\frac{d(1/\dot{x})}{dx} \tag{45}$$

$$= \left(\frac{1}{\dot{x}}\right)(\ddot{y})\left(\frac{dt}{dx}\right) + (\dot{y})\left(-\dot{x}^{-2}\ddot{x}\right)\left(\frac{dt}{dx}\right) \tag{46}$$

$$\frac{d^2y}{dx^2} = \frac{\dot{x}\ddot{y}}{\dot{x}^3} - \frac{\dot{y}\ddot{x}}{\dot{x}^3} \tag{47}$$

The second derivative of our spline is a nonlinear equation which depends on four interrelated variables. This makes bounding the radius of curvature very difficult. It is also possible to accomplish this iteratively by bounding the derivatives and solving the simultaneous equations multiple times until all values are within bounds. However both of these methods are out of the scope of this work and are not used.

4.3 Implementation

4.3.1 Communication with Khepera

The communication protocol used for the Khepera robots is very simple. Two values are sent to control the velocity of the Khepera; linear velocity and angular velocity. These are sent in a message of the following form.

Sent to Khepera: syn:ctrl:[Linear Value]:[Angular Value]:syn

The Khepera then takes this information and translates it into right and left wheel speeds. The reason for this translation was to make it appear to the program as if it were controlling a nonholonomic vehicle. The Khepera also sends a message containing battery level, Infrared distances, and Ultrasonic readings. Because of the greater accuracy of the Vicon system the data provided by this message is not used. However this data is available to use for future experiments.

Received from Khepera: ack:data:[Battery Level]:[IR Data 1-10]:[US Data 1-5]:ack

4.3.2 Path Generation Protocol

One current limitation of ROS is that message lengths for services must be static. This has caused problems when trying to send dynamic lists of waypoints and spline points. Sending only the addresses in the message is not an option, as each node is allocated memory to which other nodes do not have access. The only option is to constrain the program to a max number of waypoints, spline points, and threat locations for the path generating service. For this experiment, waypoints were limited to thirty, threats were limited to twenty-five, and the spline point count was limited to twenty thousand. These allotments are adequate for this project, especially considering the spatial constraints previously mentioned.

Sent to Path Generating Service:

Number of waypoints, X and Y data for each waypoint

Received from Path Generating Service:

Number of spline points, X and Y data for each spline point

4.4 Experimental Results

Each time the path generation is run it not only sends the location data to the robot controller but it records the sent points to a file which is then plotted in Matlab. This will allow for easy plotting of the path and will reveal if the paths do not conform to the constraints of the robots.

Quantities that we will test and measure are the ability to generate a path as close as possible to the shortest path and the ability to conform to the vehicular constraints. The first path generated is a figure eight with the ends of the loops inverted inwards. This allows us to see general differences in the characteristics of the methods. Smoothing splines tend to have sharper turns whereas B-Splines have softer curves. This property can be changed with the smoothing value in the smoothing spline; however, it decreases the importance

of reaching the waypoints. The A* algorithm is designed to take the optimal path in a discrete environment. In figure 6 it can be seen that the path does not travel in a straight line from waypoint to waypoint and this is due to discretization of the environment. This discretization error is created at angles other than 90° and will be avoided as much as possible through the tests by having the A* path travel vertically or horizontally.

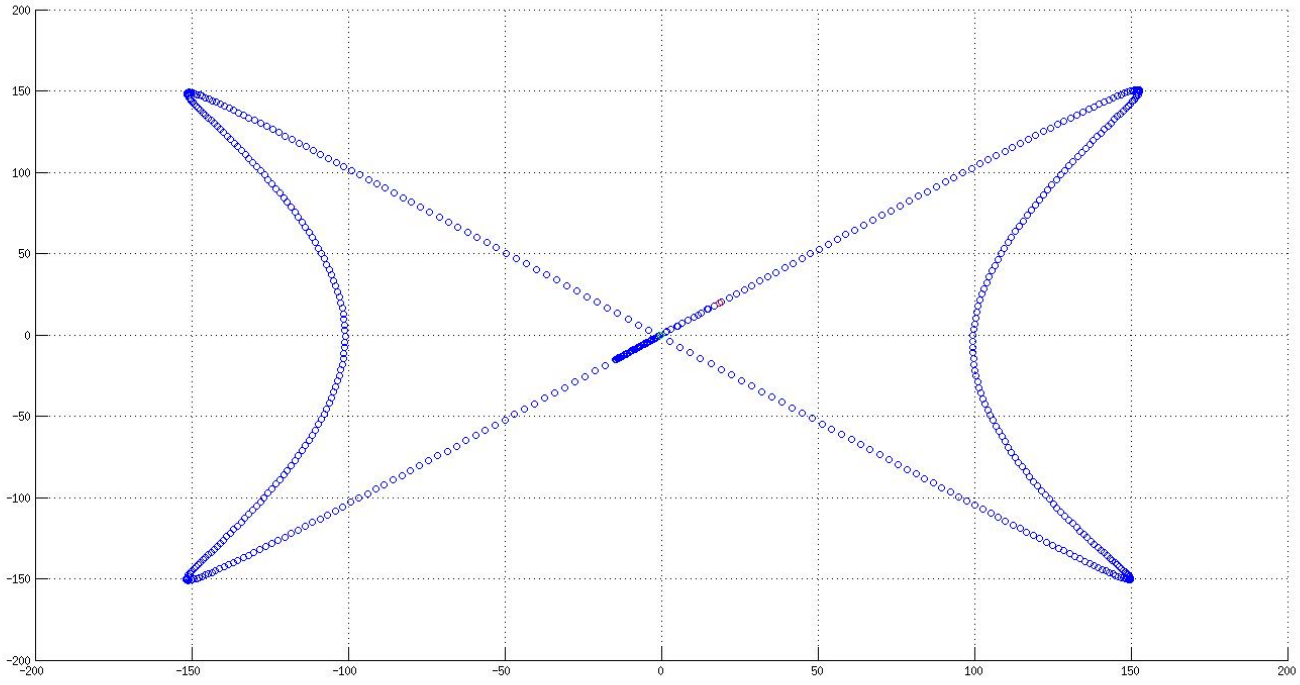

Figure 4: Example of Smoothing Spline

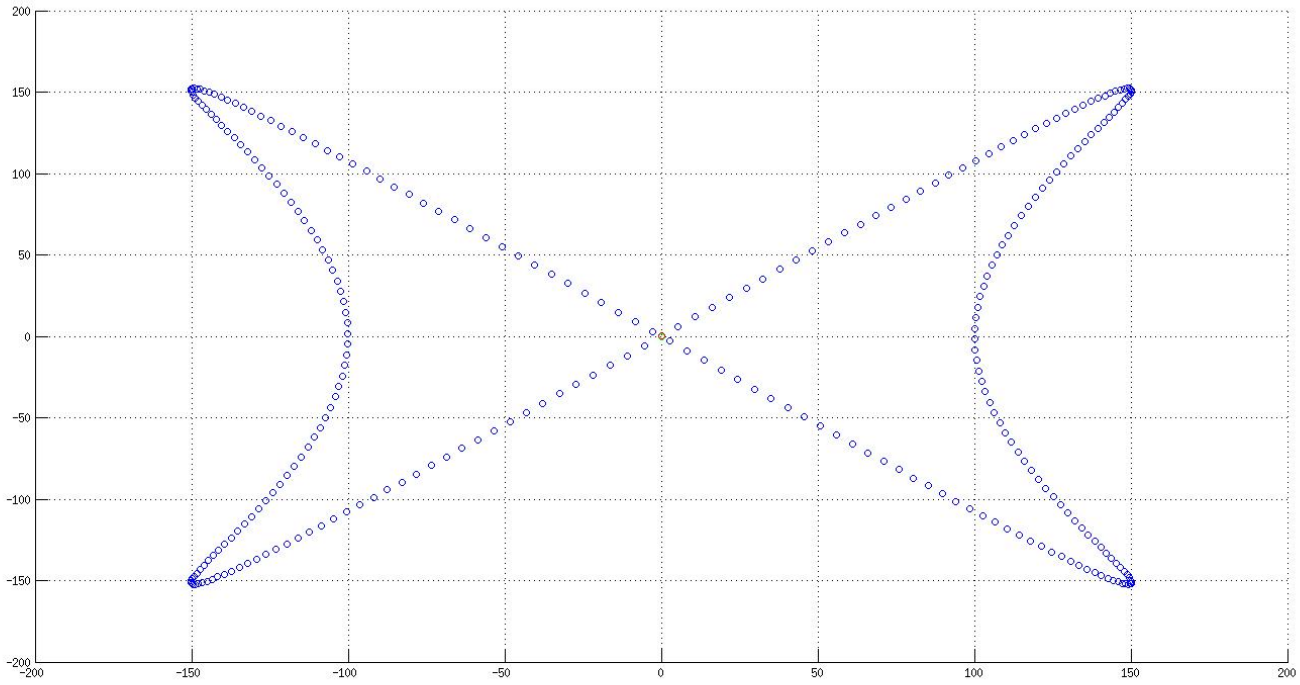

Figure 5: Example of B-Spline

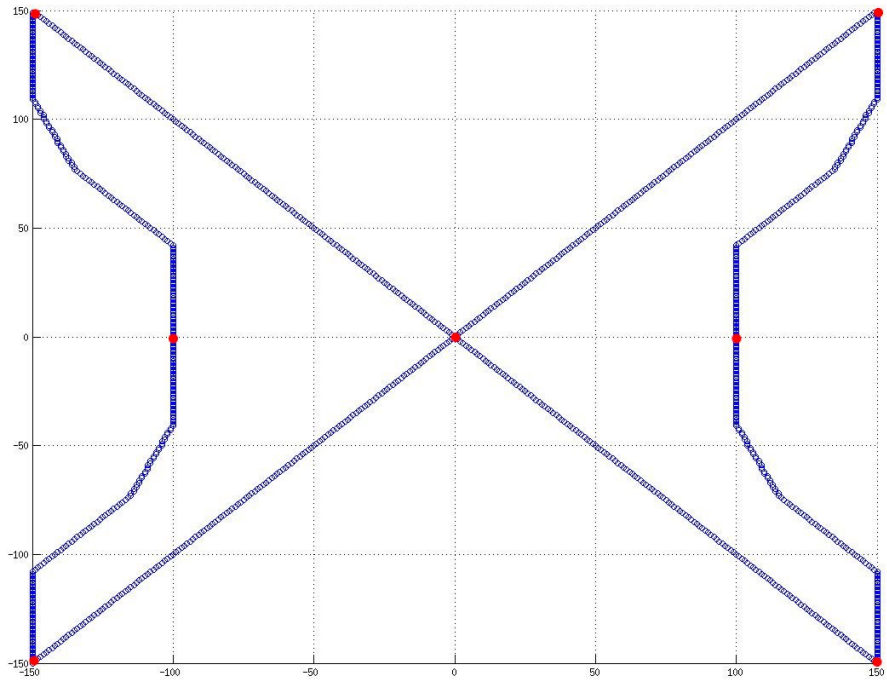

Figure 6: Example of A* search

4.4.1 Test 1: Single Turn

The first test comparing the three trajectory generation methods will be a single turn where the distance traveled will be the main measurement. The shortest path possible and still intersecting the waypoints is traversed by the Astar algorithm. This distance is compared to the B-spline and the Smoothing Spline of various smoothing values (Rho).

Table 2: Waypoints for Test 1

Waypoint	**1**	**2**	**3**
X	-150	50	50
Y	150	150	-150

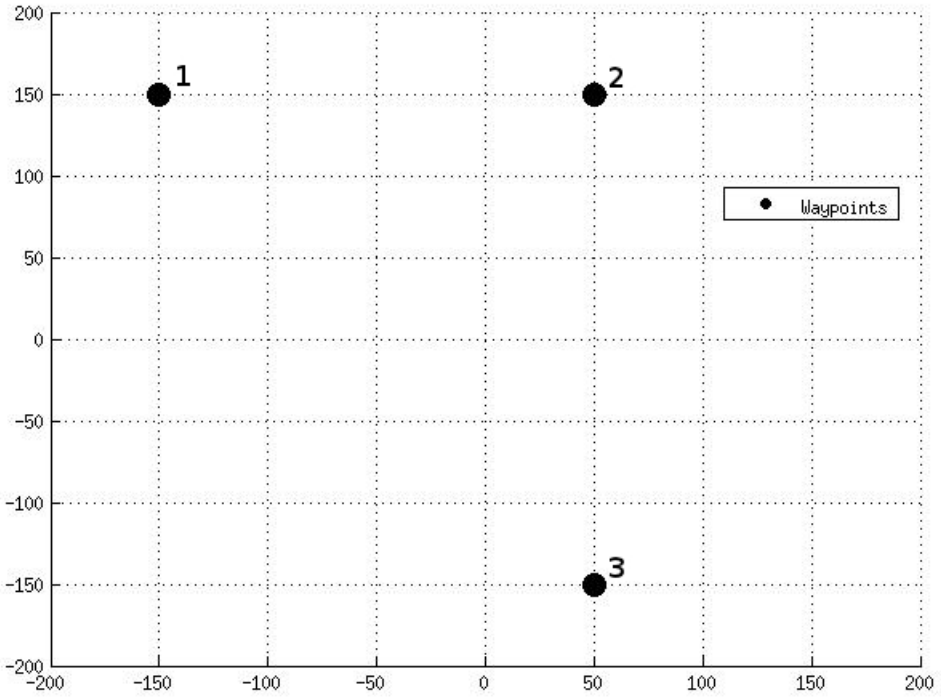

Figure 7: Waypoints for Test 1.

Table 3: Characteristics of Test 1

Method	**Distance** (cm)	**Pass Through Waypoints**	**Extra Over Baseline** (cm)	**Percent Difference**
Astar	500	✓	0	0%
B-spline	517.40	✓	17.40	3.48%
Smoothing Spline	517.90 (rho=1)	✓	17.90	3.58%
	515.21 (rho=1E6)	✓	15.21	3.04%
	495.54 (rho=1E7)		-4.46	-.89%
	417.42 (rho=1E8)		-82.58	-16.52%

Figures 8 through 11 are all three generation methods overlaid with the rho value of the smoothing spline ranging from 1 to 1×10^8. Astar generates straight lines to each waypoint and gives a distance of 500cm. B-Spline creates a path 3.48% longer but creates a path that the robot can follow without deviating from. The Smoothing Spline with rho=1 is 3.58%, this is slightly longer than the B-Spline and also intersects all the waypoints. Any rho value ranging from 1 to 1×10^6 has very little effect on the smoothness of the smoothing spline as seen in figures 8 and 9.

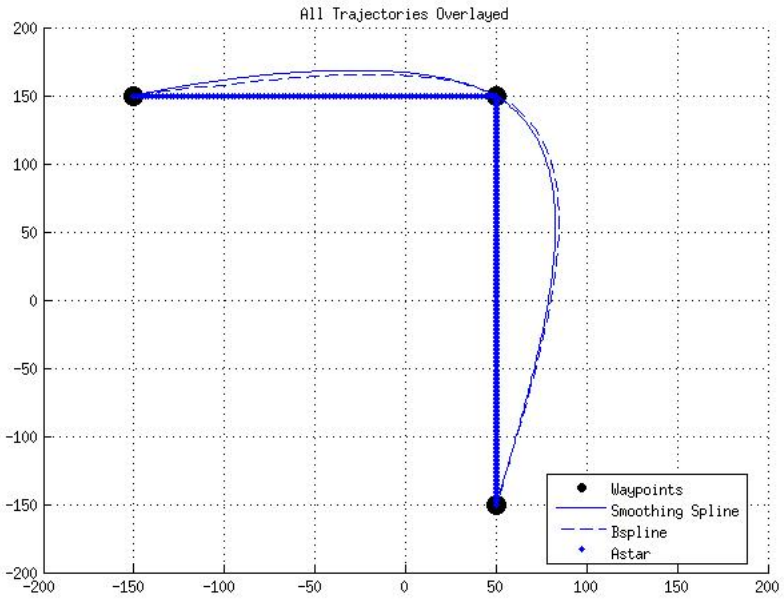

Figure 8: Large turn with rho=1 for the smoothing spline.

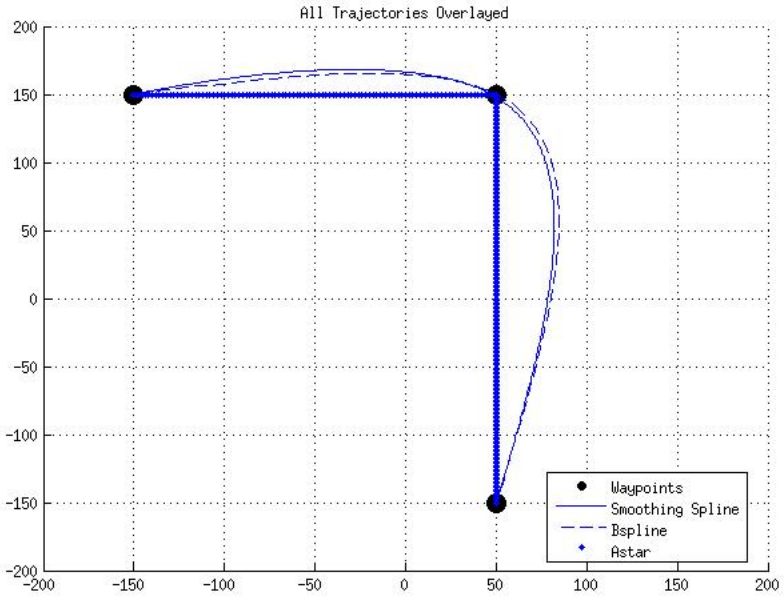

Figure 9: Large turn with rho=1E6 for the smoothing spline.

It is important to note in figure 9 the smoothing spline generates a shorter path then the B-Spline and still intersects the waypoints. This shows that if it is possible to properly tune

the spline, the smoothing spline can give a more efficient path than the B-Spline. When rho is 1×10^7 or higher the smoothing spline path no longer intersects the second waypoint, and with a value of 1×10^8 the smoothing spline no longer created a path that could be considered a valid trajectory given the set of waypoints.

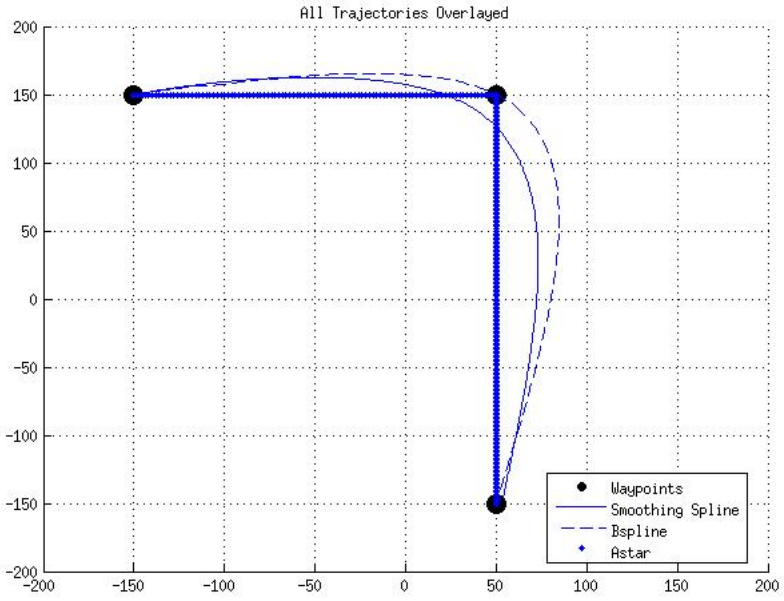

Figure 10: Large turn with rho=1E7 for the smoothing spline.

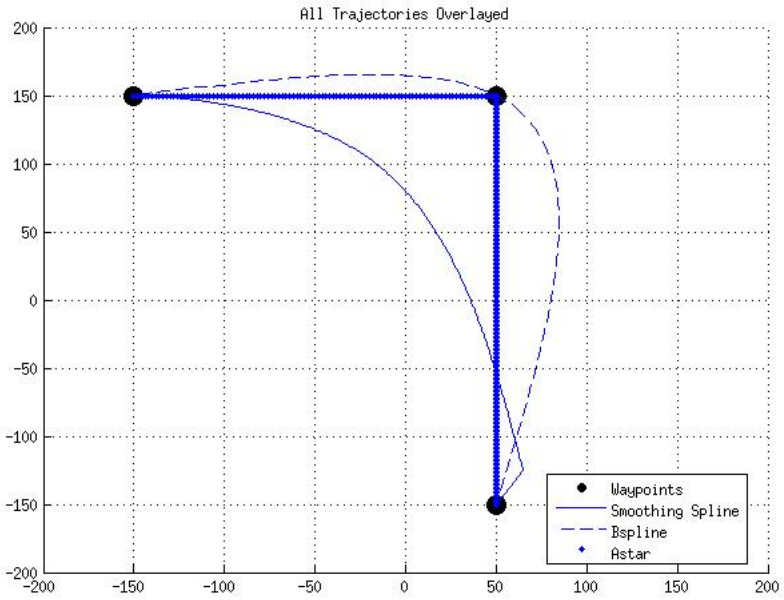

Figure 11: Large turn with rho=1E8 for the smoothing spline.

The sudden variation at the end of the smoothing spline is actually where the trajectory ends once it is calculated. However, since this is a path for a robot to traverse, we need it to reach the final waypoint. Once the robot reaches the end of the spline it will head directly to the final waypoint if the two do not align. This can be a problem for the Smoothing Spline since at higher smoothing values waypoints only influence the path and don't direct it.

4.4.2 Test 2: Small Turns

The second test is a series of waypoints comparing how the trajectories handle a long distance followed by relatively tighter turn. The large distance between each side of points is 280cm. The smaller turns increase in size starting at a distance of 5cm and increasing by 10cm, 20cm, 40cm, 80cm, and 140cm respectively for each successive turn. This test will look at each method's ability to avoid sharp corners while going around tight turns.

Table 4: Waypoints for Test 2

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
X	-140	140	140	-140	-140	140	140	-140	-140	140	140	-140	-140	140
Y	145	145	140	140	130	130	110	110	70	70	-10	-10	-150	-150

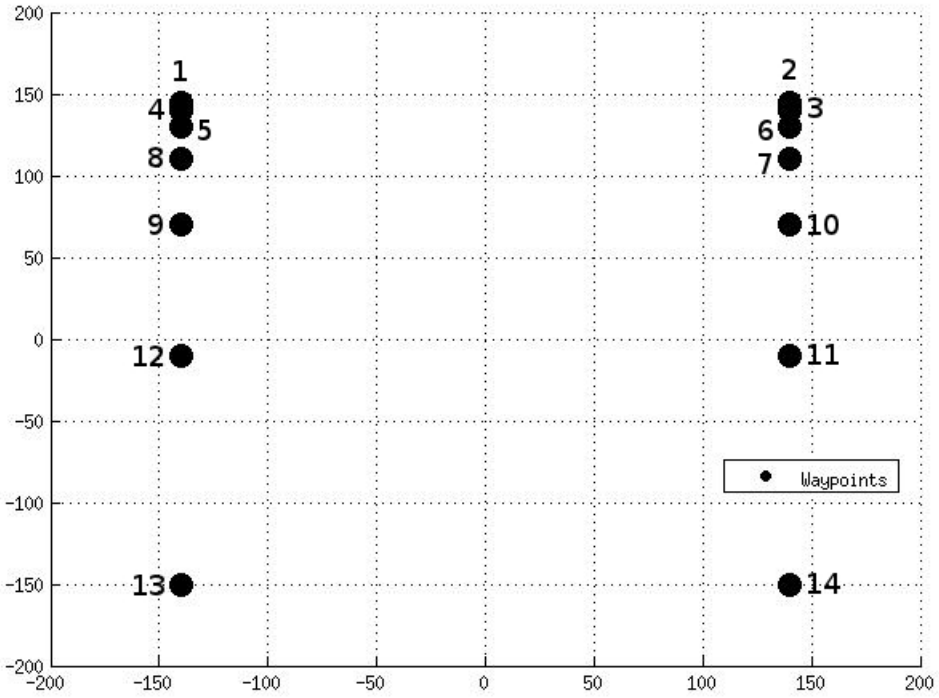

Figure 12: Waypoints for Test 2.

Table 5: Characteristics of Test 2

Method	**Distance** (cm)	**Pass Through Waypoints**	**Extra Over Baseline** (cm)	**Percent Difference**	**Figure**
Astar	2255	✓	0	0%	13
B-spline	2465.64	✓	210.64	9.34%	14
Smoothing Spline	2780.86 (rho=1)	✓	525.86	23.32%	15
	2780.87 (rho=1E3)	✓	525.87	23.32%	16
	2771.46 (rho=1E5)	✓	516.46	22.90%	17
	2690.84 (rho=1E6)		435.84	19.33%	18
	2143.93 (rho=1E7)		-111.07	-4.93%	19
	909.61 (rho=1E8)		-1345.386	-59.66%	20

Figure 13 is the path created by the Astar algorithm. It traverses the waypoints with a total distance of 2255cm. This path will only be useful if the resolution of the map is greater than the turning radius of the robot. If the robot can turn 90° within one cell of the Astar map, then it will work.

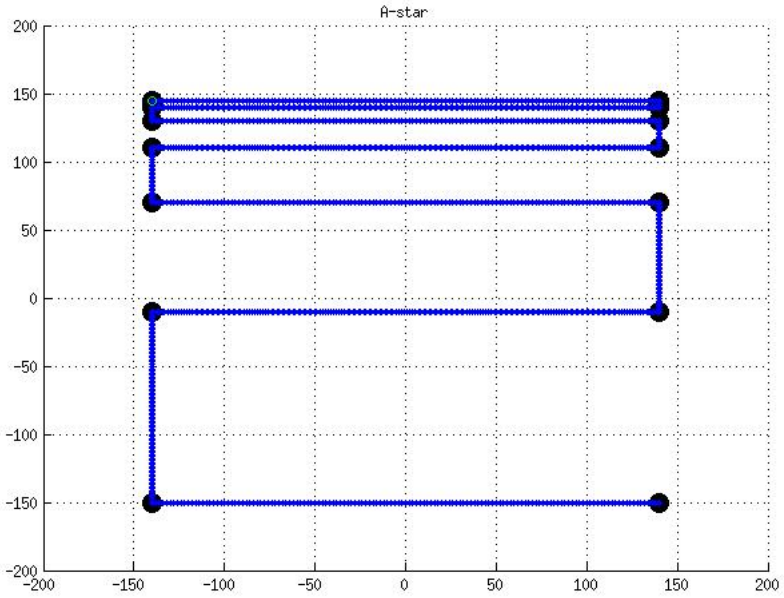

Figure 13: Astar generated path.

However, if the trajectory needs to take into consideration for the turn radius of the robot, the B-Spline would be a more appropriate choice. The curvature of the path allows for the robot to stay on track without overshooting the waypoints. This is because the B-Spline method is taking into consideration the next three points to guide the angle of entry.

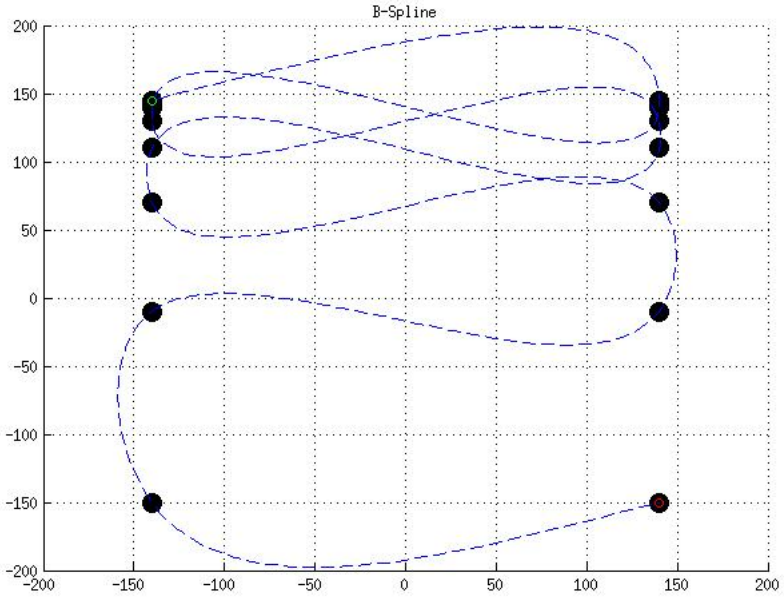

Figure 14: B-Spline generated path.

In contrast to the B-Spline, the Smoothing Spline overshoots the waypoints if they are too close together. Figure 15 is an example of how the smoothing spline overshoots the waypoints. The way to compensate for this in the smoothing spline is to increase the rho value which "smooths" out the path. Figures 16 through 20 show increased rho values and the effect it has on the generated path.

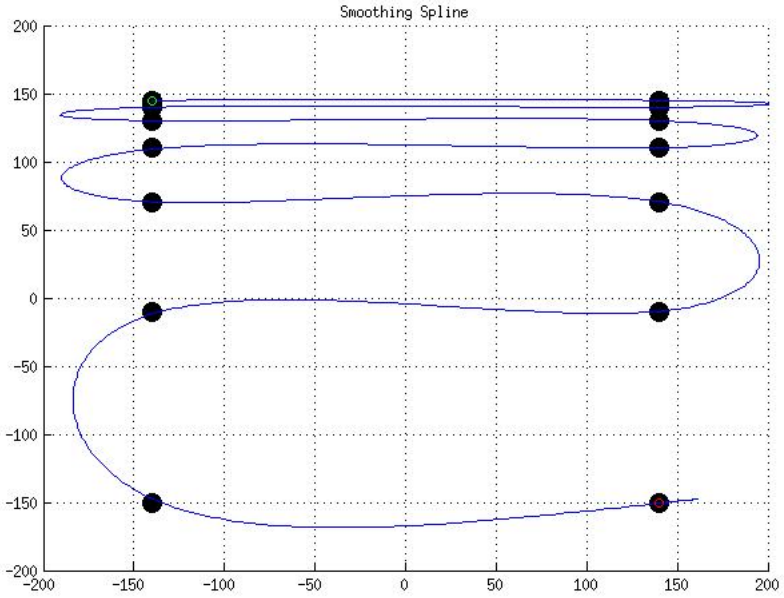

Figure 15: Smoothing Spline with rho=1.

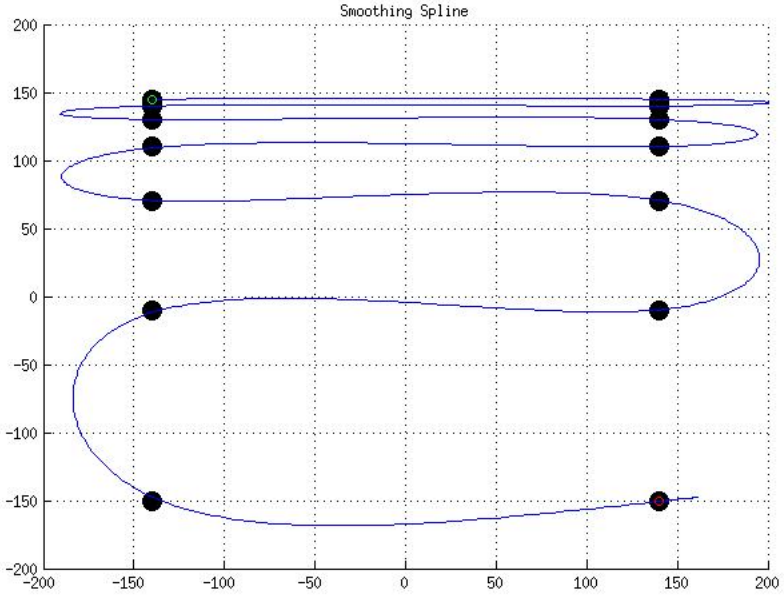

Figure 16: Smoothing Spline with rho=1E3.

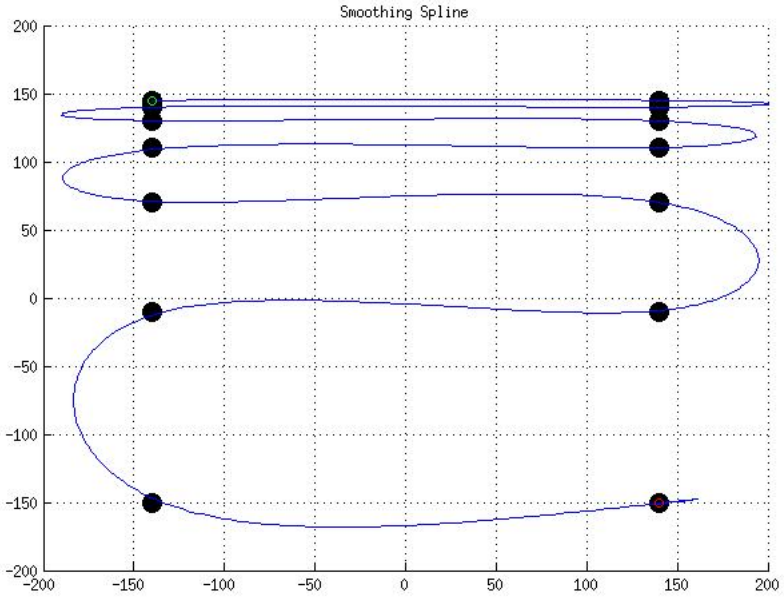

Figure 17: Smoothing Spline with rho=1E5.

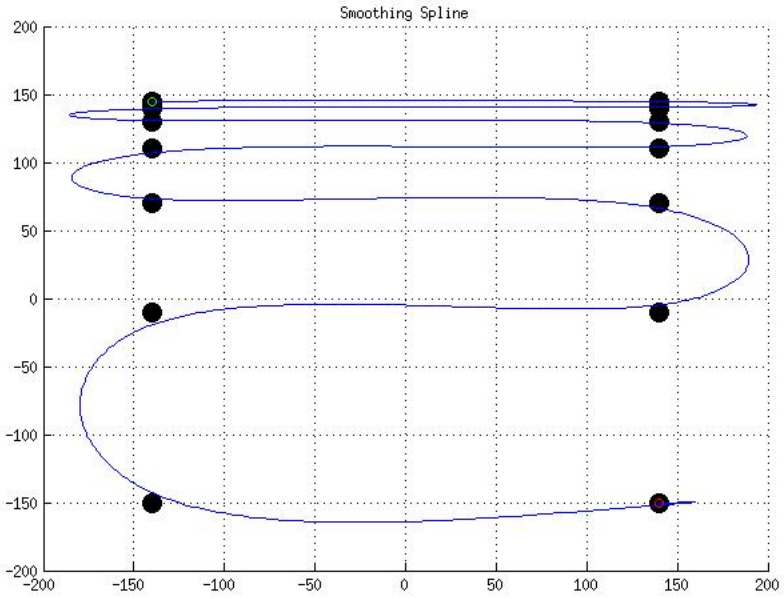

Figure 18: Smoothing Spline with rho=1E6.

In Figure 18 with a rho value of 1×10^6 the spline is starting to have a smaller overshoot on the waypoints that are closed together. The problem is that this rho value is also causing

the spline to pull away from the waypoints in the larger turns. This trend continues into higher rho values. It is possible to avoid this problem with additional tuning and weights that will cause different waypoints to affect the sections of the spline differently.

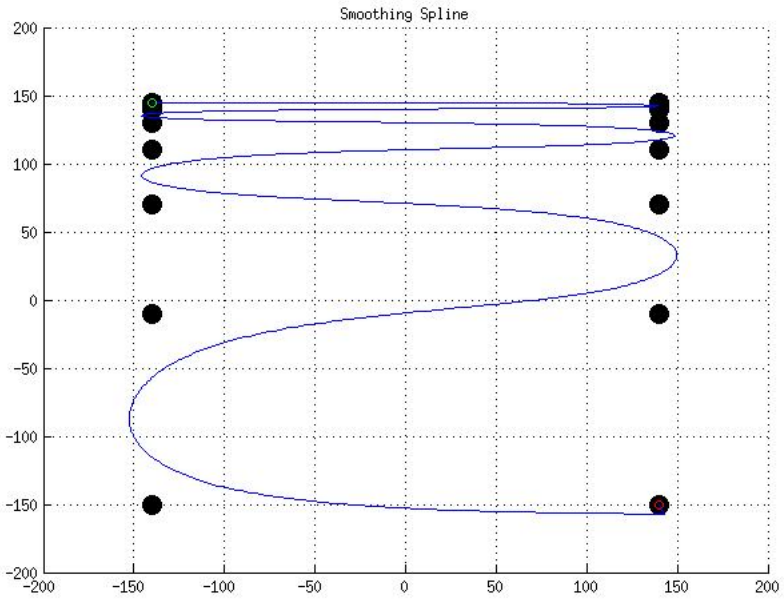

Figure 19: Smoothing Spline with rho=1E7.

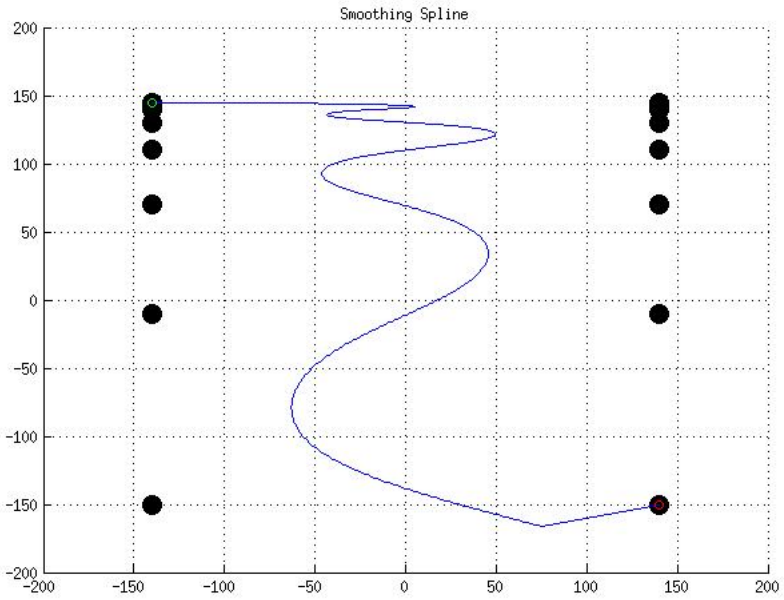

Figure 20: Smoothing Spline with rho=1E8.

With rho=1×10^7 the spline has eliminated the overshoot but has pulled away from most waypoints and would no longer be considered arriving at these destinations. Once rho has reached 1×10^8 the path is no longer acceptable by any means. The conclusion reached from this test is B-Spline has a greater ability to avoid sharp turns that would cause the robot to deviate from the path. B-Spline also generates a path which is much closer to the Astar baseline than any of the acceptable Smoothing Spline paths. Smoothing Splines cannot adapt to waypoints that are close together.

4.4.3 Test 3: Circle

This test involves a series of four repeating waypoints. While the Astar method merely makes a square connecting each of the points, the splines create a circle. The waypoints repeat multiple times so the spline trajectories have time to stabilize to a circular pattern. This test allows us to see the effect of rho on the smoothing spline and how far it causes the spline to pull away from the waypoints. It also shows minor characteristic differences between the B-Spline and Smoothing Spline. The waypoints start in the upper left corner, repeat four times, then end in the bottom right corner.

Table 6: Waypoints for Test 3

Waypoints	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
X	-100	100	100	-100	-100	100	100	-100	-100	100
Y	100	100	-100	-100	100	100	-100	-100	100	100
Waypoints	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	
X	100	-100	-100	100	100	-100	-100	100	100	
Y	-100	-100	100	100	-100	-100	100	100	-100	

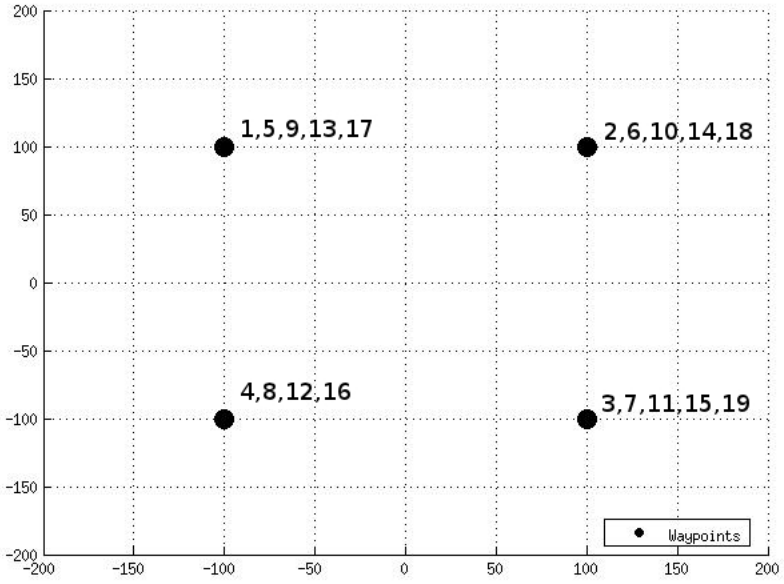

Figure 21: Waypoints for Test 3.

Table 7: Characteristics of Test 3

Method	**Distance** (cm)	**Pass Through Waypoints**	**Extra Over Baseline** (cm)	**Percent Difference**	**Figure**
Astar	3600	✓	0	0%	23
B-spline	3918.39	✓	318.39	8.84%	24
Smoothing Spline	3961.07 (rho=1)	✓	361.07	10.03%	25
	3951.97 (rho=1E5)	✓	351.97	9.78%	27
	3870.81 (rho=1E6)		270.81	7.52%	29
	3242.02 (rho=1E7)		-357.98	-9.94%	31
	1385.86 (rho=1E8)		-2214.14	-61.50%	33

With all three methods overlapped it can be seen that the two spline trajectories quickly stabilize to approximately the same circular path.

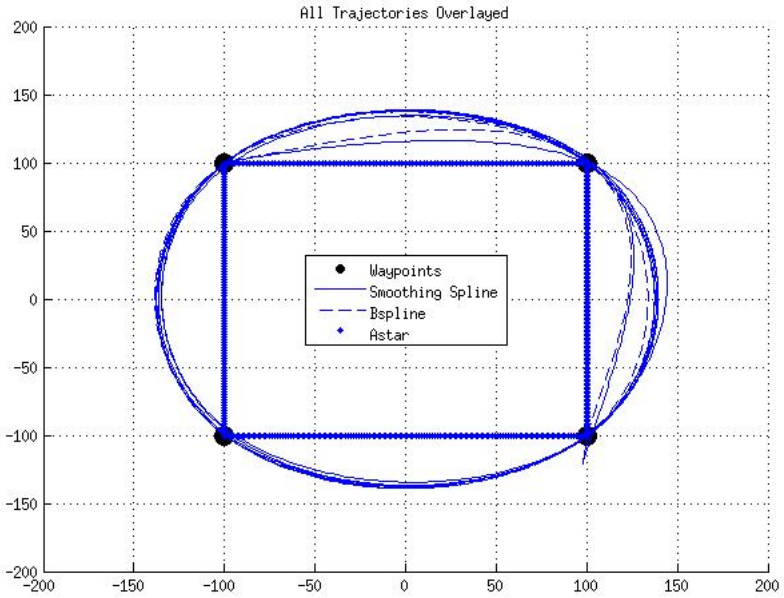

Figure 22: Circle with rho=1 for the smoothing spline.

The Astar trajectory creates a path that is 800cm per loop with a total distance traveled of 3,600cm.

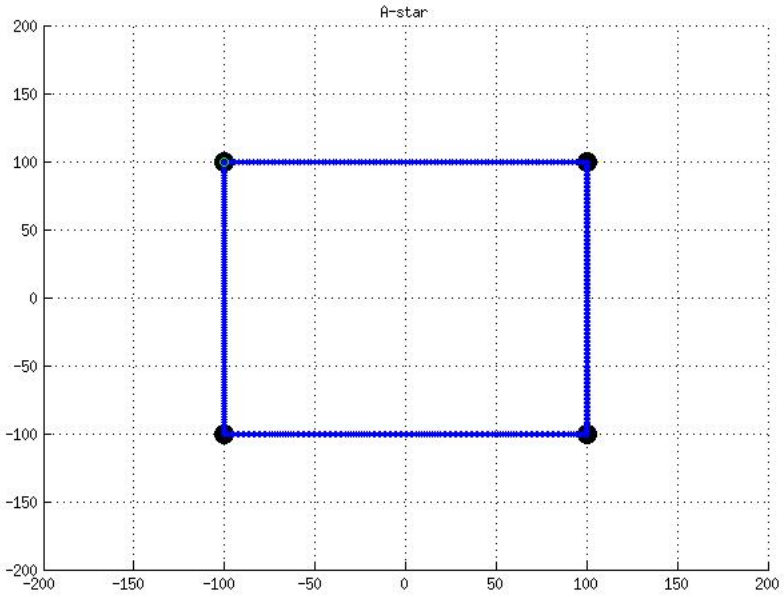

Figure 23: Astar following waypoints.

Figure 24 shows the B-Spline traversing the given set of waypoints. In the first and second segment the spline is slowly getting closer to the stable circular pattern. By the third segment the spline has stabilized as a result of being based on third order polynomials.

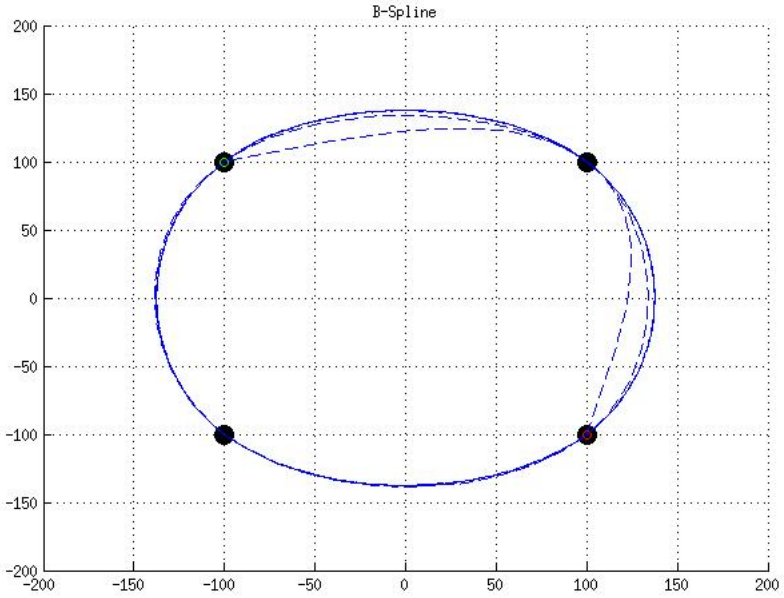

Figure 24: B-Spline.

Unlike the B-Spline the Smoothing Spline never reaches a consistent pattern because each spline point is based on every point before it. This can be seen clearly in figure 26 as none of the smoothing spline cycles overlap each other.

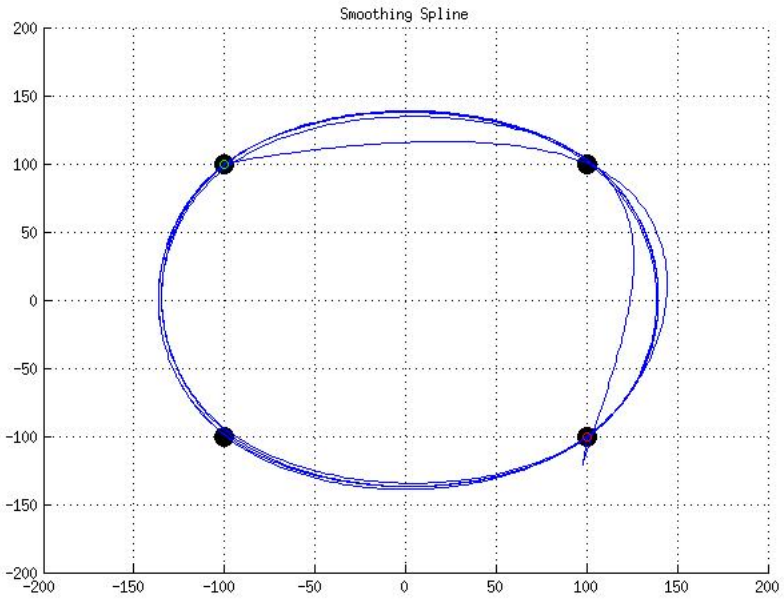

Figure 25: Smoothing Spline with rho=1.

Each of the four smoothing spline loops are slightly different whereas the B-Spline paths in this corner of the circle are following the same path.

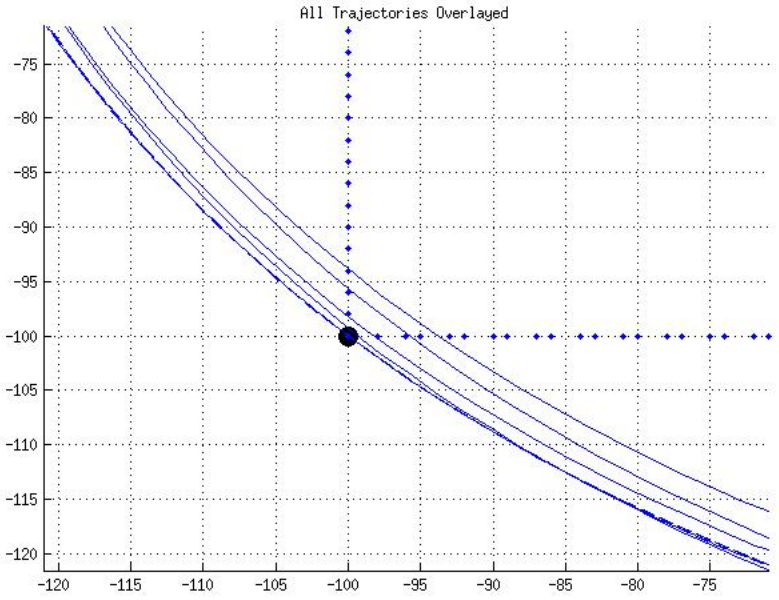

Figure 26: Bottom left corner with rho=1.

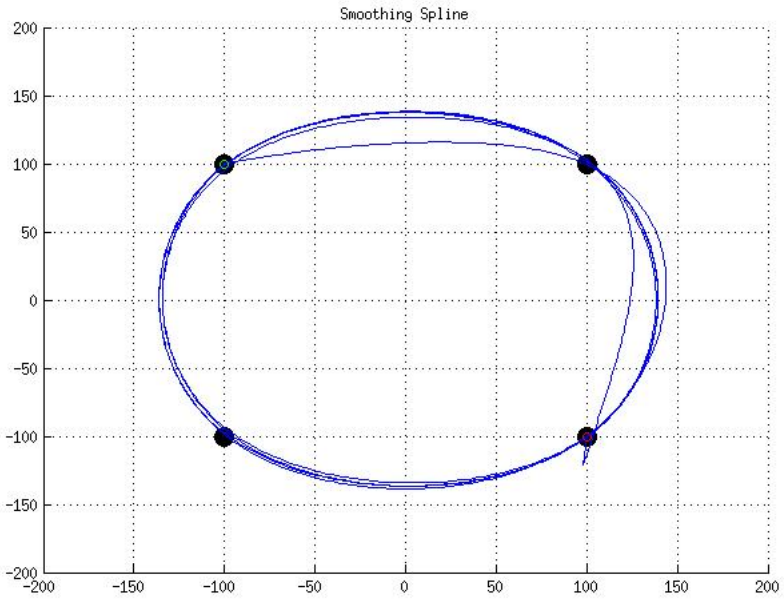

Figure 27: Smoothing Spline with rho=1E5.

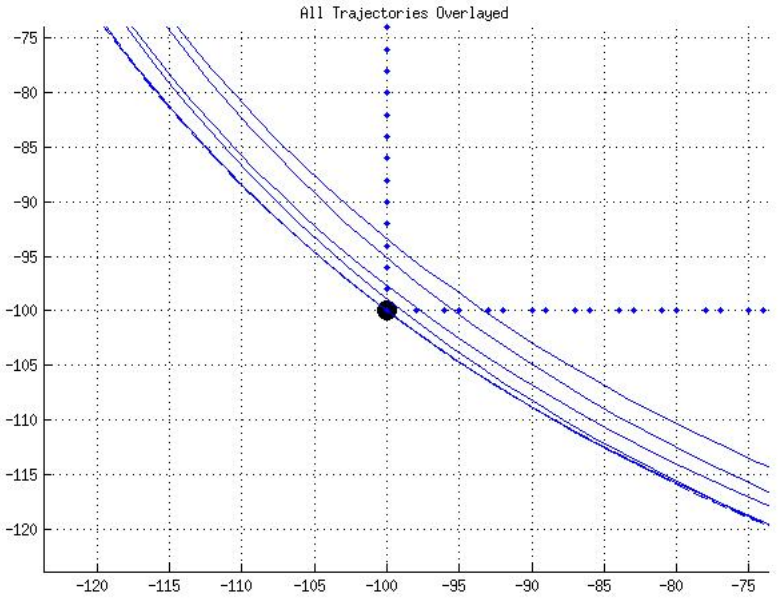

Figure 28: Bottom left corner with rho=1E5.

With rho=1×10^6 the smoothing spline paths are pulling away from each of the waypoints. In these experiments intersecting the waypoint is considered to be less than 5cm

from the center of the waypoint. As shown in figure 30, most smoothing spline paths are not within the 5cm radius to the waypoint which is considered the cutoff for intersecting the waypoint.

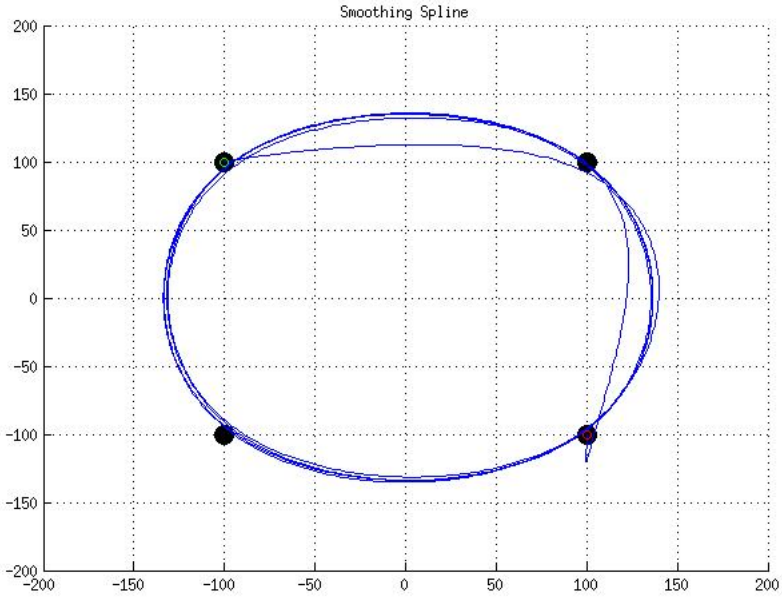

Figure 29: Smoothing Spline with rho=1E6.

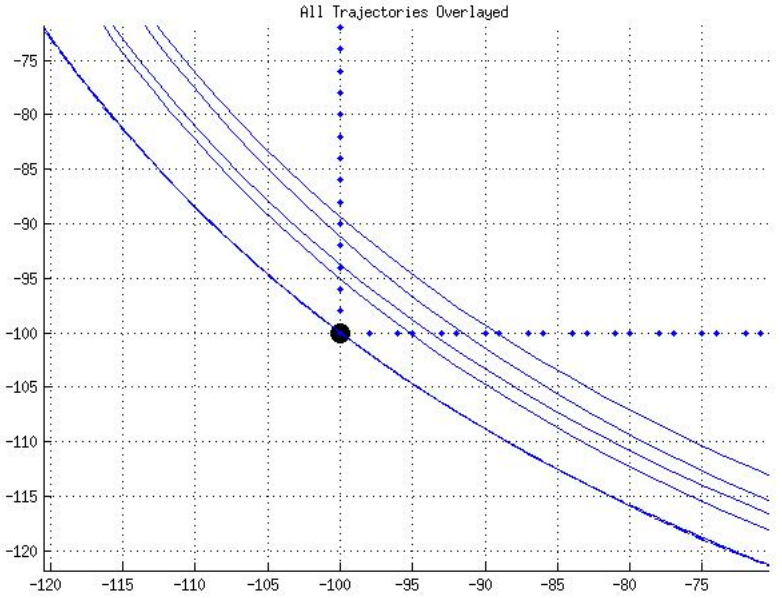

Figure 30: Bottom left corner with rho=1E6.

As rho increases further to 1×10^7 each waypoint has less of an effect on the path of the spline and as a result it is no longer getting close to any of the waypoints.

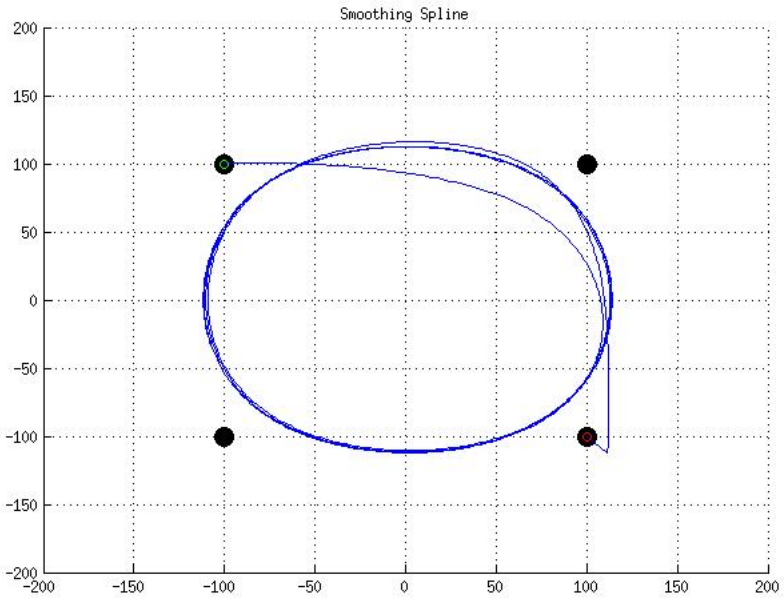

Figure 31: Smoothing Spline with rho=1E7.

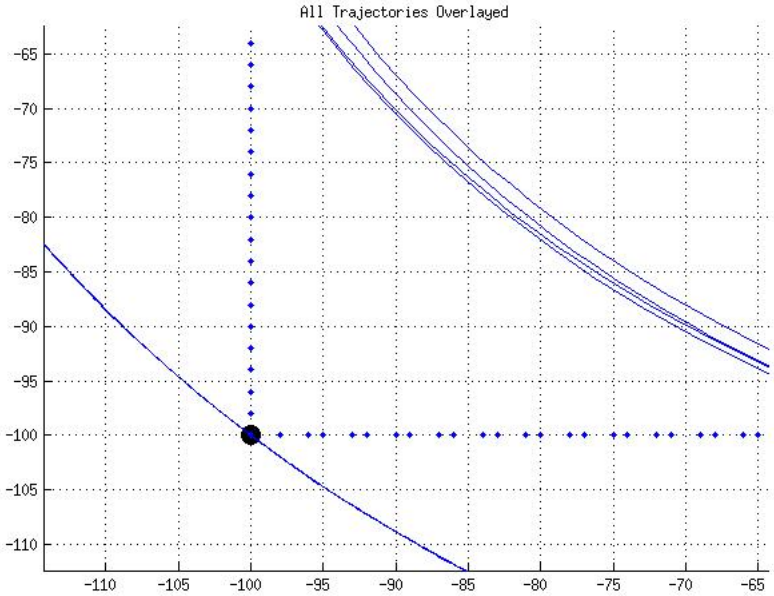

Figure 32: Bottom left corner with rho=1E7.

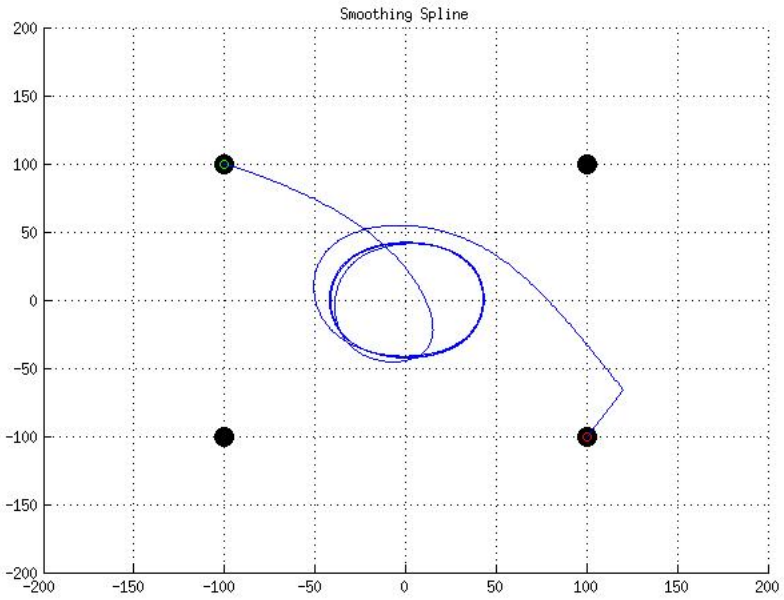

Figure 33: Smoothing Spline with rho=1E8.

As in the first test there is a point to which the smoothing spline can be tuned and be slightly more efficient than the B-Spline, but by default the B-Spline creates a path that is relatively efficient with no tuning. Since the waypoints in this test are far enough apart, the robots should avoid swinging out as far as the splines would normally demand. It would still be within the dynamics of the robots to create straighter paths to each waypoint. This highlights a problem of efficiency with the splines as these paths are scaled up to miles.

4.4.4 Test 4: Multiple Trips between Two Points

In situations where multiple trips need to be made between two points the path generation methods will have to completely reverse their heading to go to the next waypoint. The radius which is maintained for turning is the main point of this study.

Table 8: Waypoints for Test 4

	1	**2**	**3**	**4**	**5**	**6**
X	150	150	-150	150	-150	150
Y	150	-150	150	-150	150	-150

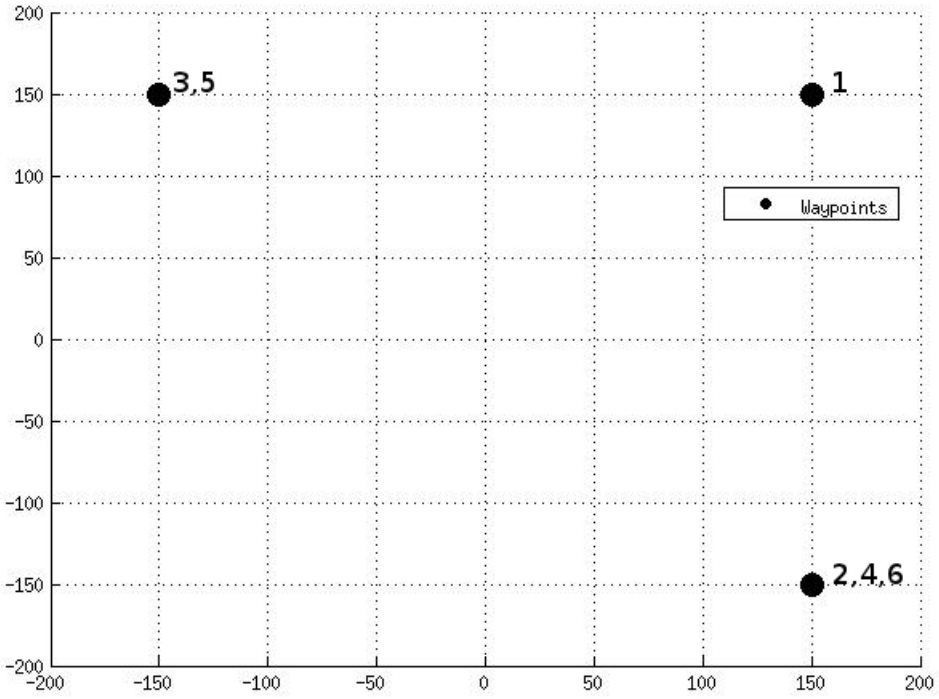

Figure 34: Waypoints for Test 4.

Table 9: Characteristics of Test 4

Method	**Distance** (cm)	**Pass Through Waypoints**	**Extra Over Baseline** (cm)	**% Difference**	**Figure**
Astar	1997.06	✓	0	0%	35
B-spline	2045.81	✓	48.75	2.44%	37
Smoothing Spline	2066.59 (rho=1)	✓	69.53	3.48%	35
	2063.82 (rho=1E5)	✓	66.76	3.34%	38
	2039.27 (rho=1E6)	✓	42.21	2.11%	40
	1824.93 (rho=1E7)		-172.13	-8.62%	42
	1004.34 (rho=1E8)		-992.72	-49.71%	44

The first waypoint starts in the top right and goes back and forth between the other two waypoints twice. The first time is to see what kind of radius results with an initial heading not directed strait at the next waypoint. The second iteration is to observe the turn radius when a complete 180° turn is needed.

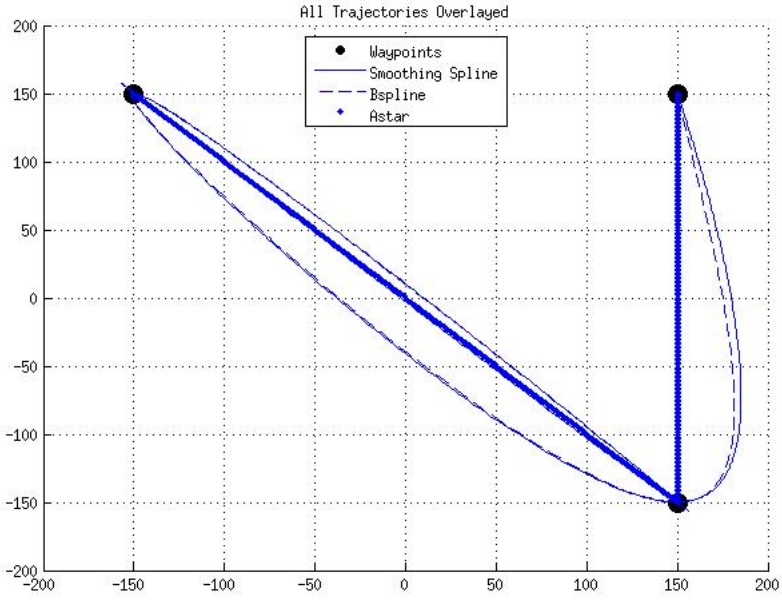

Figure 35: Back and Forth, All methods, rho=1.

Figure 36 is a close-up of the top end point showing how the first loop of each spline has a tolerable curvature but once they come back for a second pass, they each overshoot the waypoint and create a turn which is not compatible with the robot dynamics.

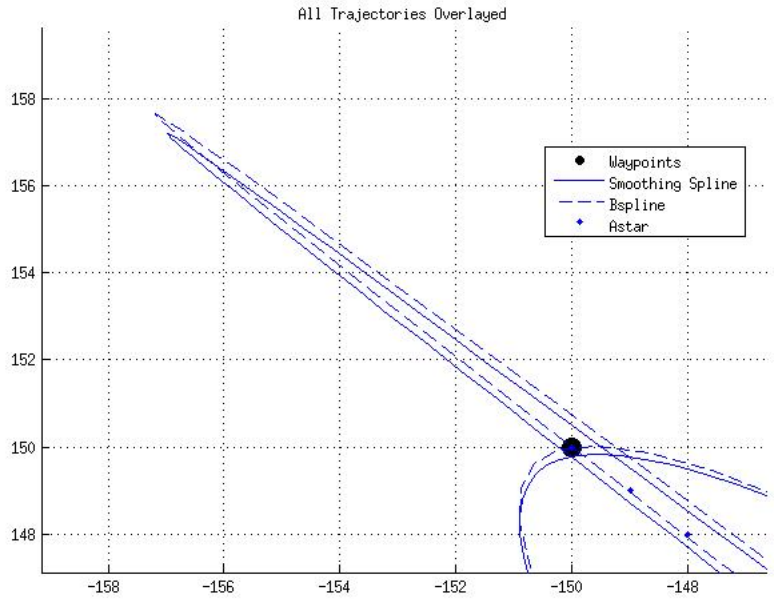

Figure 36: End Point, All Methods, rho=1.

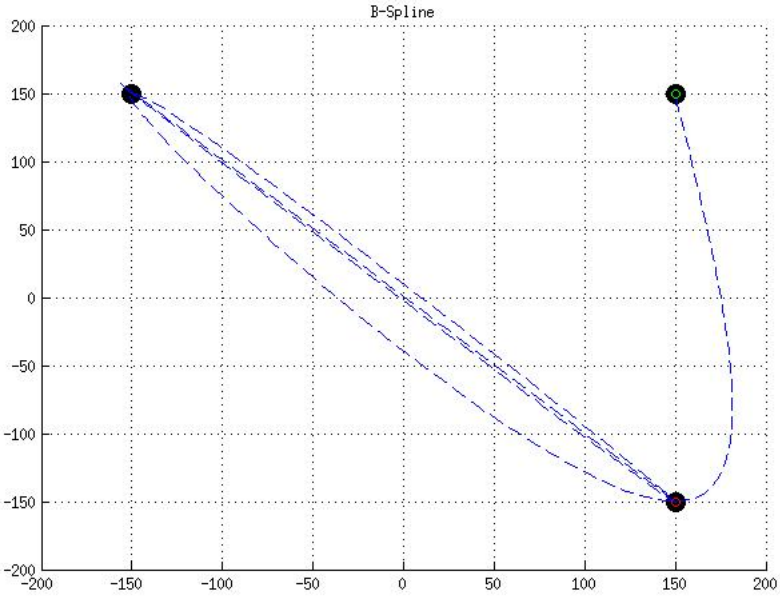

Figure 37: B-spline.

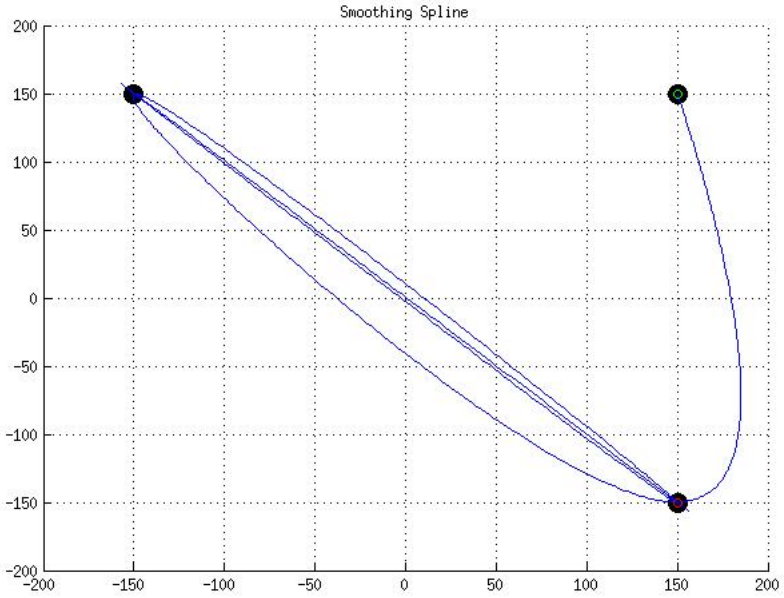

Figure 38: Smoothing Spline, rho=1E5.

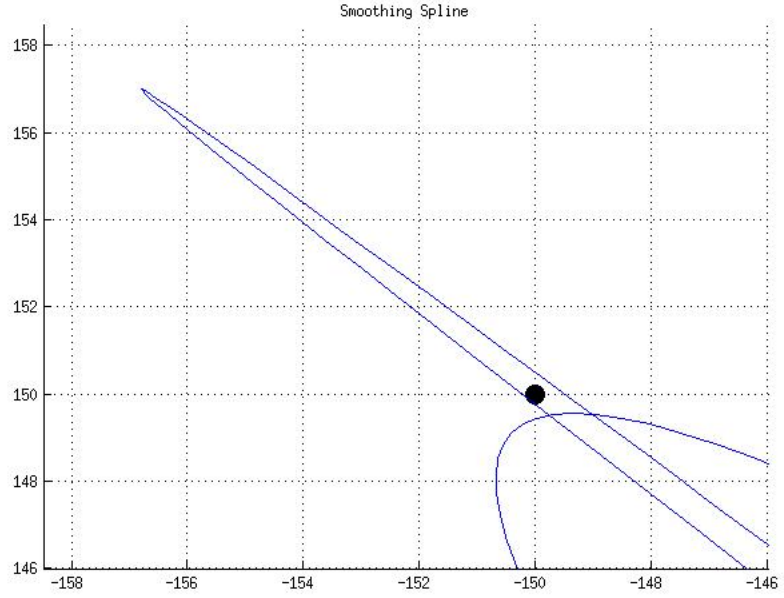

Figure 39: Smoothing Spline Corner with rho=1E5.

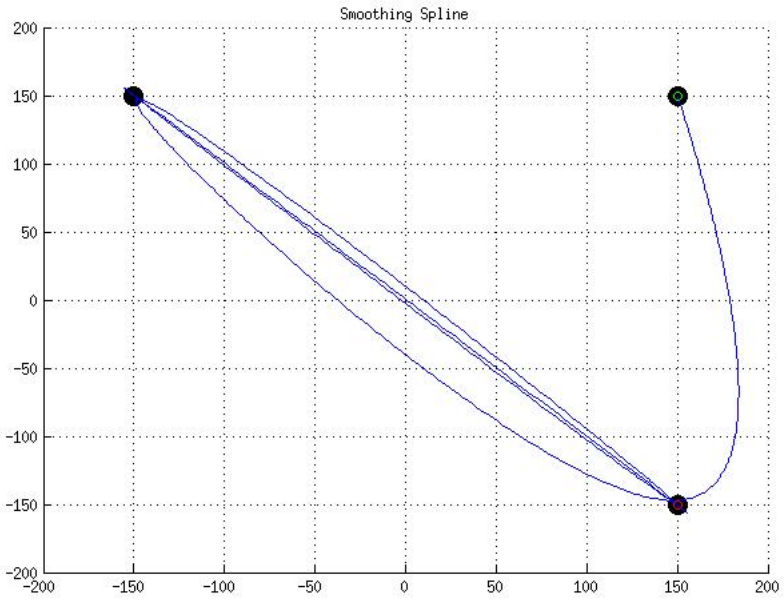

Figure 40: Smoothing Spline, rho=1E6.

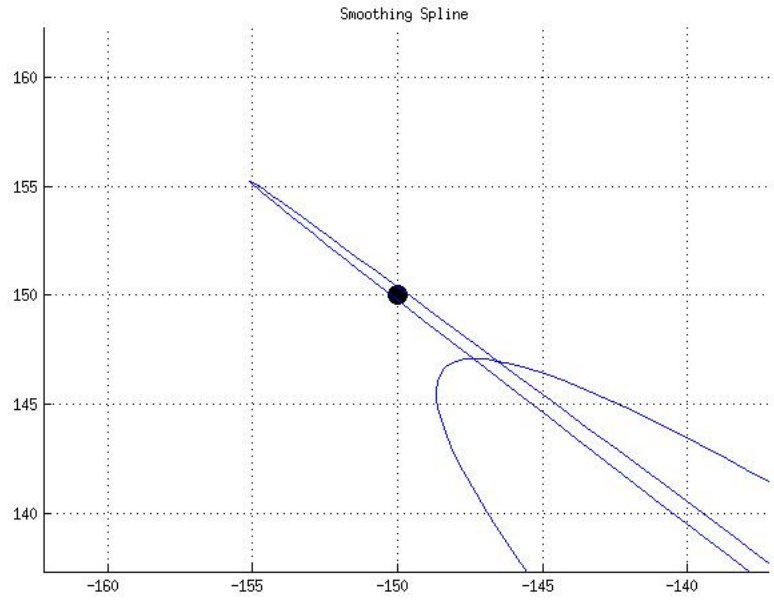

Figure 41: Smoothing Spline Corner with rho=1E6.

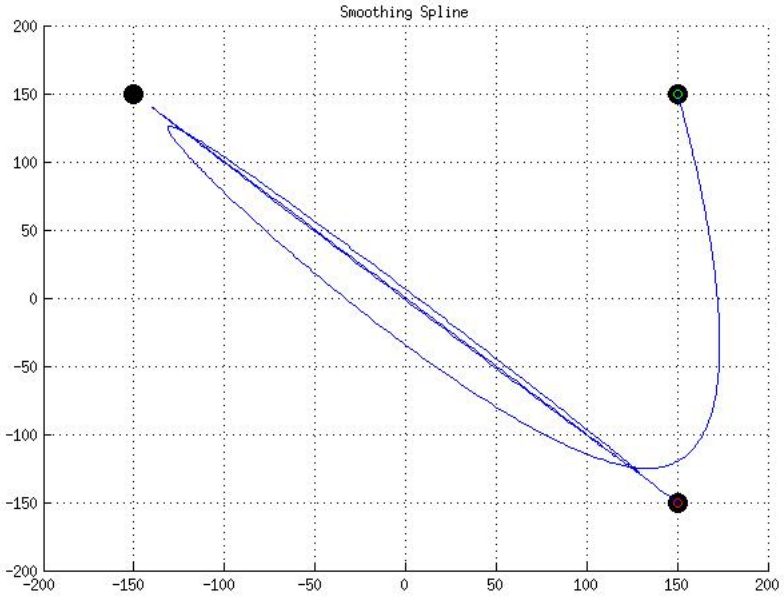

Figure 42: Smoothing Spline, rho=1E7.

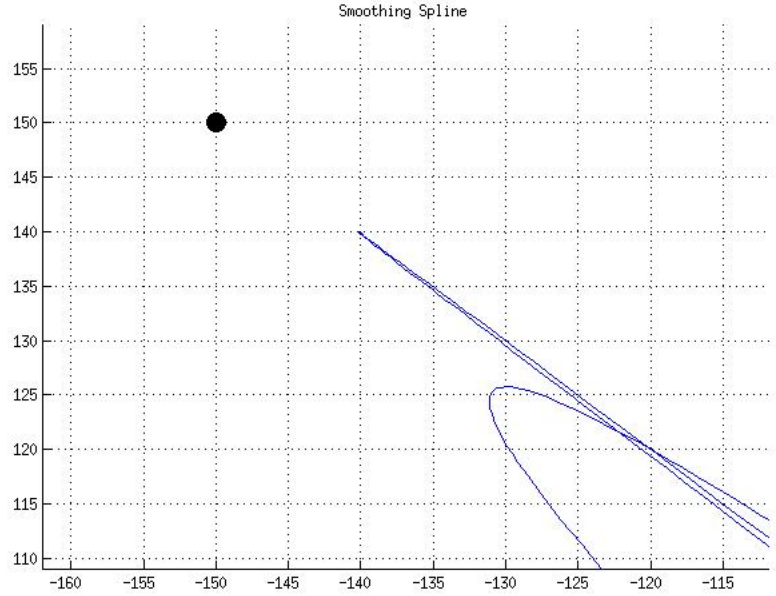

Figure 43: Smoothing Spline Corner with rho=1E7.

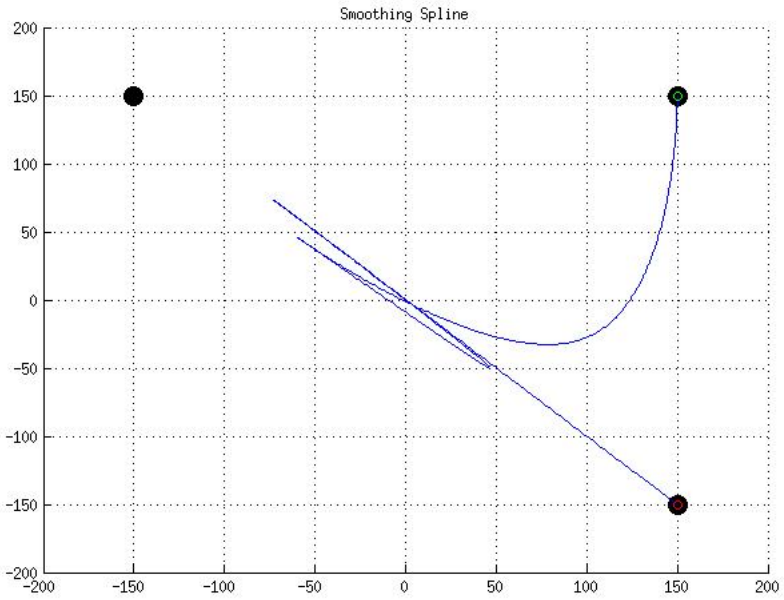

Figure 44: Smoothing Spline, rho=1E8.

This test shows that none of these methods are very good in the described situation. All three methods generate a path that is similar in style, none of which fit the dynamics of the robot. Even though up to this point the B-Spline has created smooth turns that fit within the dynamics of our vehicle, this test shows that a sharp corner is still possible if the waypoints are repeating or overlapping. Even though the B-Spline is taking into consideration the next three waypoints, those waypoints form a straight line folded back on itself, thus the generated spline is nearly a straight line.

4.4.5 Test 5: Few Obstacles

The purpose of this test is to observe the relative ability of each method to avoid obstacles. The Astar method is assigned an infinite cost for the threat regions and results in a path that avoids threats. However B-Splines and Smoothing Splines have no inherent way to avoid threats and thus must have multiple iterations if a threat zone is intersected. Multiple things can be done within these iterations including reshaping the path to go around the threat, or introduce new waypoints and regenerate the path.

Table 10: Waypoints for Test 5

Waypoint	**1**	**2**	**3**
X	-150	100	50
Y	150	100	-150

Table 11: Threats for Test 5

Threat Zones	**1**	**2**	**3**
X	20	40	75
Y	130	0	-100
Diameter	40	80	30

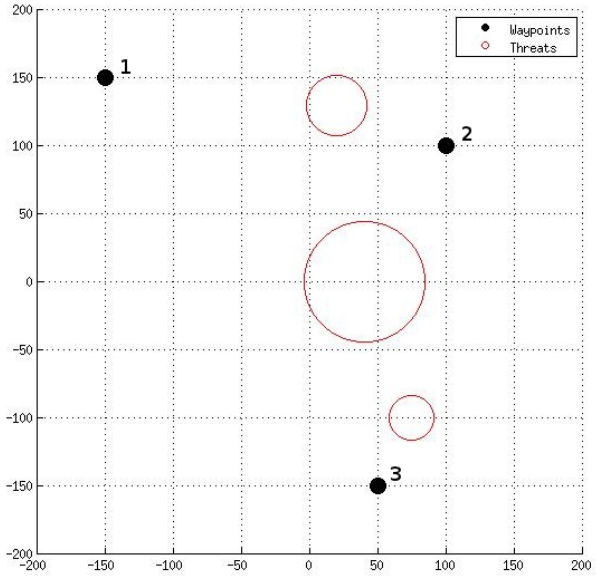

Figure 45: Waypoints and threat zones.

In this test multiple iterations were taken of the B-Spline to examine how adding waypoints around the threat zones affect the overall generated path. It is important to note that simply driving around the path creates a longer path than planning ahead and taking the threat zones into consideration. This planning ahead can reduce the extra distance by more than half.

Table 12: Characteristics of Test 5

Method	**Distance** (cm)	**Pass Through Waypoints**	**Extra Over Baseline** (cm)	**% Difference**	**Figure**
Astar	523.607	✓	0	0%	46
B-Spline	559.73	✓	36.12	6.90%	47
	539.96 (2nd iteration)	✓	16.36	3.12%	48
	540.04 (3rd iteration)	✓	16.44	3.14%	49
Smoothing Spline	587.47 (rho=1)	✓	63.87	12.20%	50
	579.53 (rho=1)	✓	55.92	10.68%	51
	585.14 (rho=1E6)	✓	61.54	11.75%	52
	577.479 (rho=1E6)	✓	53.87	10.29%	53
	558.241 (rho=1E7)		34.63	6.61%	54
	535.86 (rho=1E7)		12.25	2.34%	55

The Astar algorithm always takes the shortest discretized path after taking into consideration threat zones as obstacles. In all of the following graphs the dashed line represents the path taken by the trajectory method while the bold line represents the adjusted path after taking into account the threats. With Astar both lines are the same since the Astar algorithm is able to take in the positions of the threats

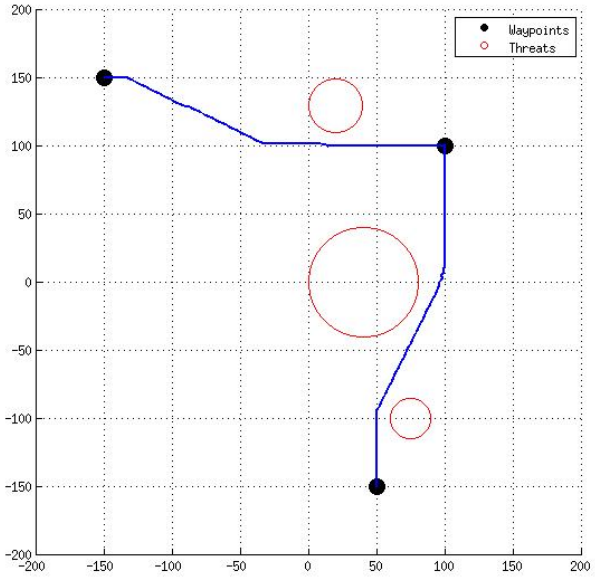

Figure 46: Astar.

With the first iteration of the B-Spline algorithm the spline passes through two of the target zones. One possibility is to push the spline points just outside of the threat radius as seen in figure 47.

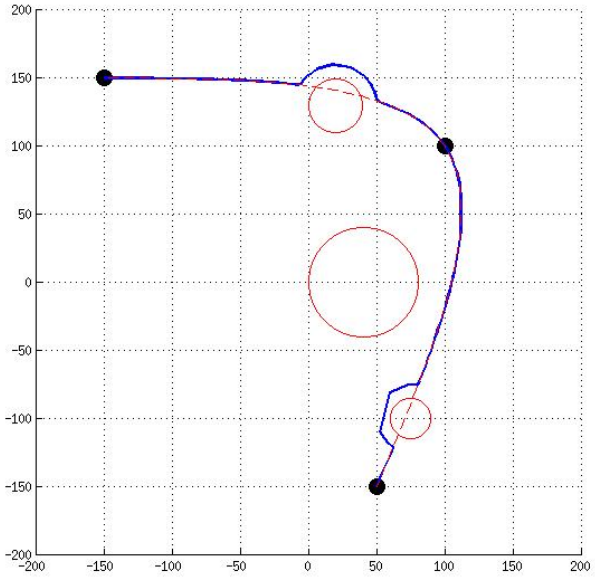

Figure 47: B-Spline with one iteration.

One of the issue with simply pushing spline outside of the threat radius is that it may create a sudden turn that are outside the dynamics of the robot. One simple solution is to add a waypoint just outside the threat and recalculate the path. These multiple iterations are shown in figures 48 and 49.

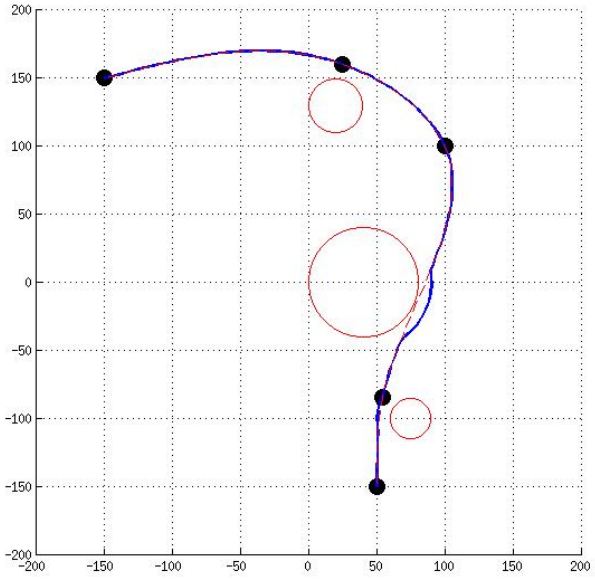

Figure 48: B-Spline second iteration with additional waypoints.

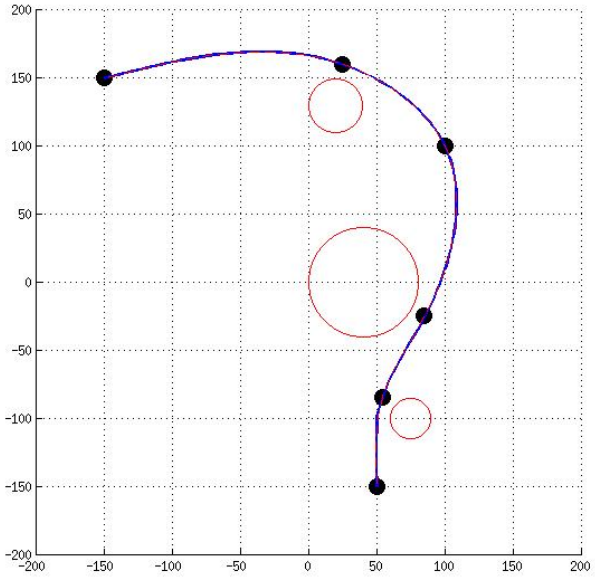

Figure 49: B-Spline third iteration with additional waypoints.

The same thing that was done with the B-Spline is done with the smoothing spline now. The first step is to run the smoothing spline algorithm and push the spline points outside the threat radius. Then waypoints are placed where the spline was pushed out and the algorithm is run again.

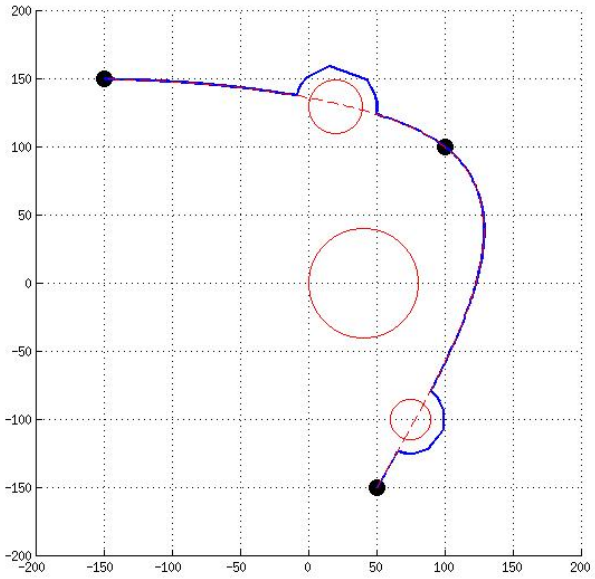

Figure 50: Smoothing Spline, rho=1, first iteration.

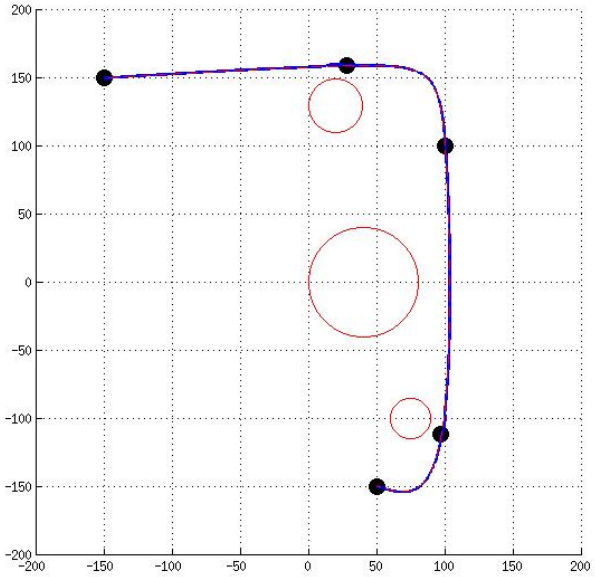

Figure 51: Smoothing Spline, rho=1, second iteration.

Adding waypoints to avoid threats also worked with rho=1×10^6.

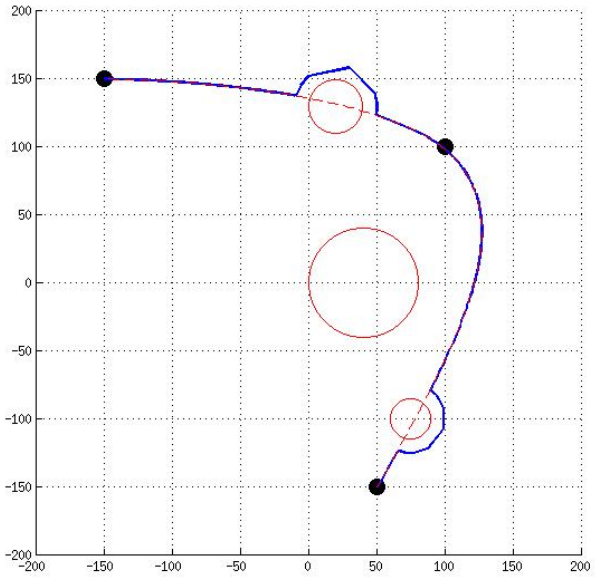

Figure 52: Smoothing Spline, rho=1E6, first iteration.

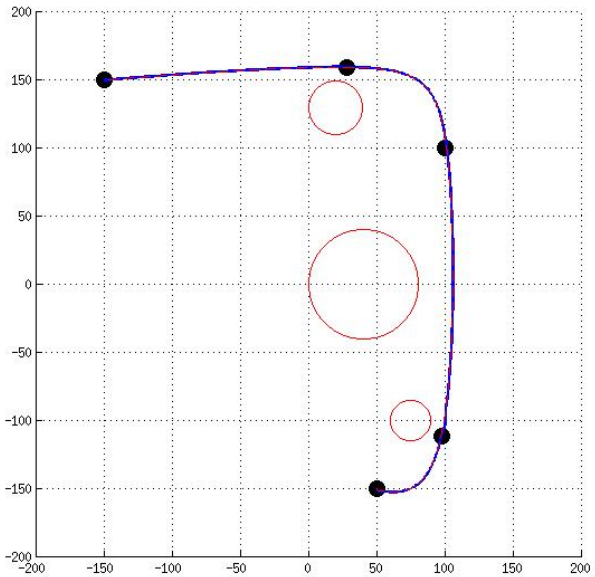

Figure 53: Smoothing Spline, rho=1E6, second iteration.

In the case of the Smoothing Spline where rho=1×10^7, no single waypoint had enough of an effect to pull the generated path out of the threat radius as can be seen in figure 54 and

55. With the smoothing factor being the main benefit of the Smoothing Spline there is no reason to use this method of additional waypoints to avoid threat zones. There is no way to judge how strong of an effect the additional waypoint will have on the path or if it will be enough to pull it out of the threat zone.

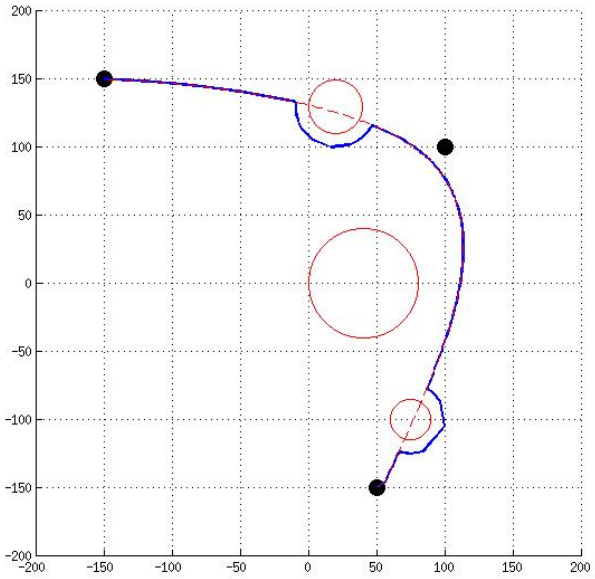

Figure 54: Smoothing Spline, rho=1E7, first iteration.

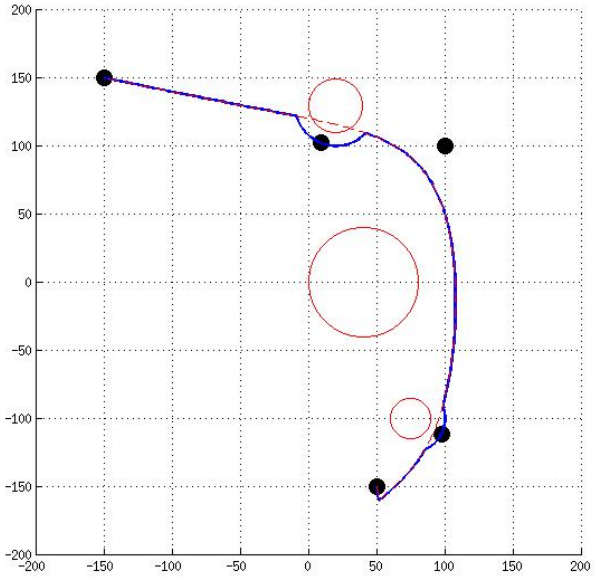

Figure 55: Smoothing Spline, rho=1E7, second iteration.

In conclusion the method of adding waypoints to avoid threat zones helps in simple environments with a sparse distribution of threats. It also provides shorter paths if the threats are known in advance as opposed to driving around the outside edge of the zone. With the only advantage of Smoothing Splines over B-Splines being the smoothing parameter, B-Splines are a better choice if waypoints are used to avoid threats.

4.4.6 Test 6: Many Obstacles

The last test shows how the three methods were able to manage maneuvering around a small number of threats. This test will look into how each method is able to cope if there is a relatively large number of threats. The testing area is filled with miscellaneous sized and placed threat zones that the trajectories must maneuver around.

Table 13: Waypoints for Test 6

Waypoint	**1**	**2**	**3**	**4**	**5**
X	-150	0	100	50	-150
Y	150	0	100	-150	-50

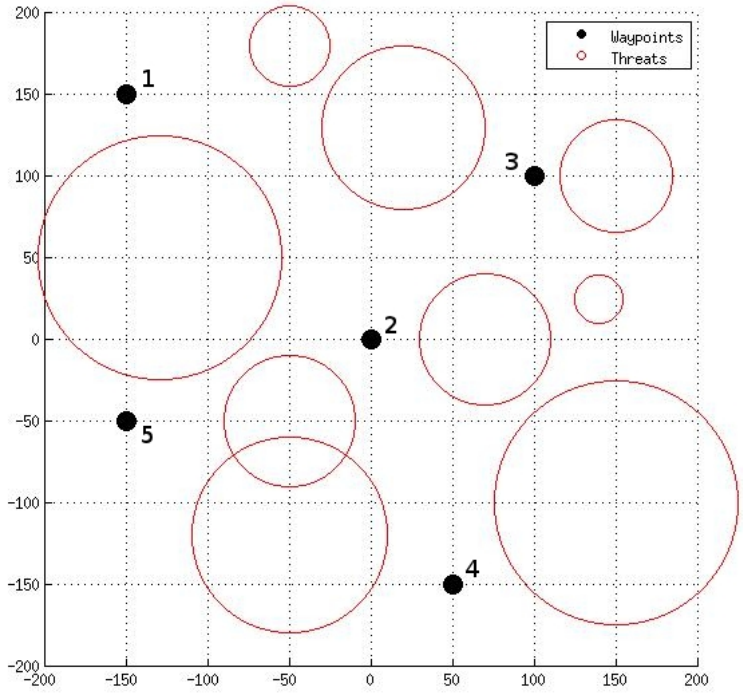

Figure 56: Waypoints and threat zones

Table 14: Characteristics of Test 6

Method	**Distance** (cm)	**Pass Through Waypoints**	**Extra Over Baseline** (cm)	**% Difference**	**Figure**
Astar	952.41	✓	0	0%	57
B-Spline	1003.75	✓	51.34	5.39%	58
Smoothing Spline	990.70 (rho=1)	✓	38.29	4.02%	59
	966.46 (rho=1E6)	✓	14.06	1.48%	60
	821.56 (rho=1E7)		-130.85	-13.74%	61

Astar is able to find the shortest path possible while remaining a certain distance away from threat zones. This type of environment is where the Astar algorithm shows its strengths. It is able to find the shortest path to each waypoint while avoiding all the threats.

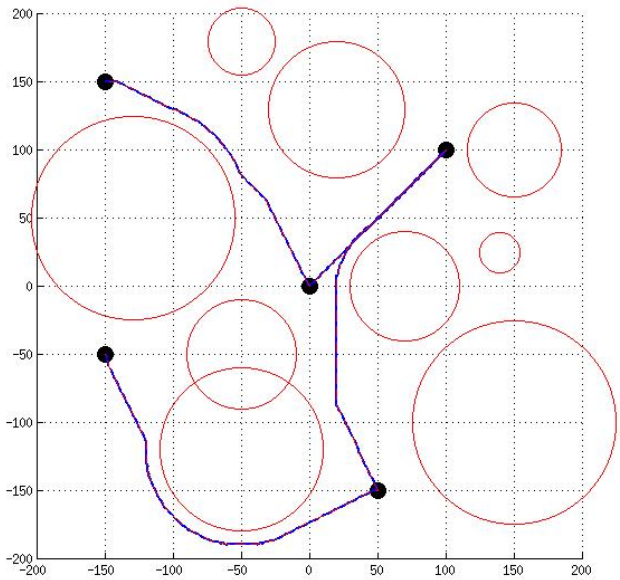

Figure 57: Astar

The B-Spline takes the waypoints and plots its path as normal, but as with the previous test, parts of the spline must be moved to avoid threats. The problem this creates is the

spline might be pushed into another threat zone. This can lead to an infinite loop or the robot getting stuck in a local minimum depending on how the threats are avoided. An example of this can be seen in figure 61. The route of the B-Spline between the third and fourth waypoint is not wide enough to pass through. This is why the Astar algorithm did not pass through this shorter route.

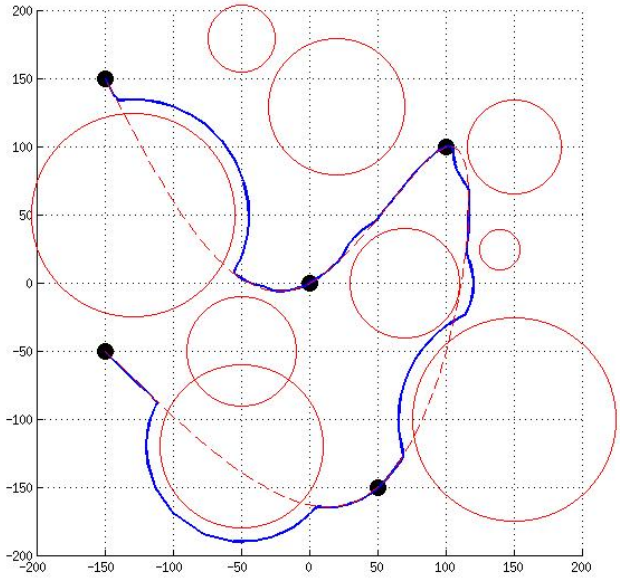

Figure 58: B-Spline

Smoothing splines are a worse choice than B-Splines for this kind of environment. The entire point of a smoothing spline is to have a variable smoothing parameter that can be adjusted. This determines the proximity of the splines to the waypoints. In the case where there are few or no threats this can work; however in the case of many threats there is no way to tell if the spline not reaching the waypoint will adversely affect the resulting path.

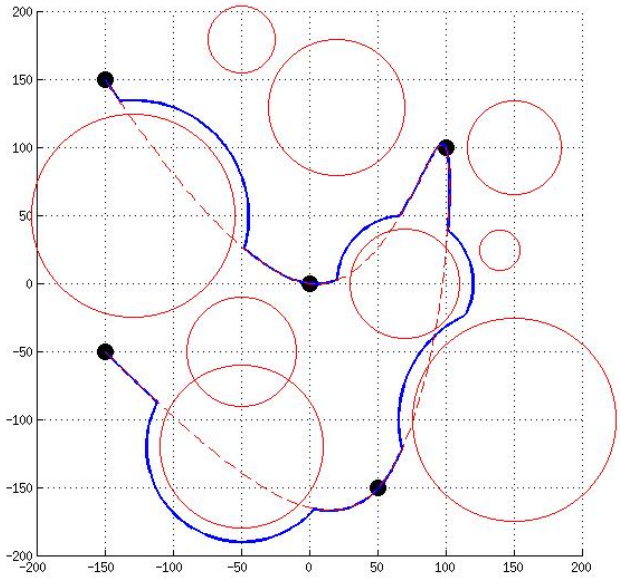

Figure 59: Smoothing Spline rho=1

Even if more waypoints were used to have the splines avoid the additional threats this would only make things worse and is another reason why splines cannot be used in such an environment. The additional waypoints could be generated inside a threat zone and result in an infinite loop or be generated in such a way that makes the generated path very undesirable.

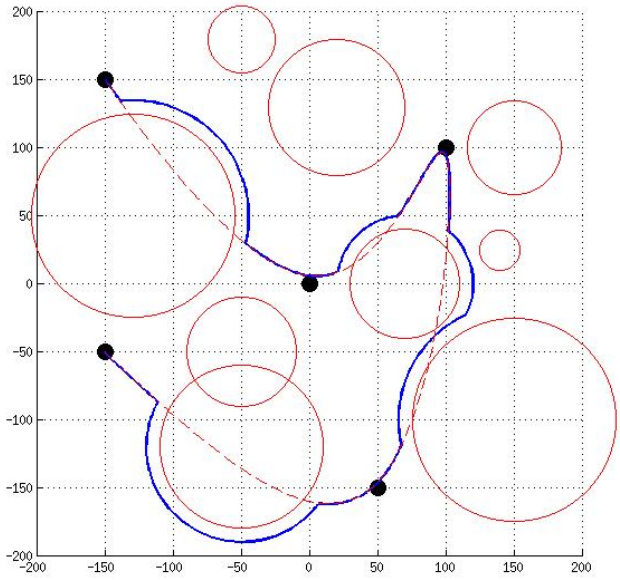

Figure 60: Smoothing Spline rho=1E6

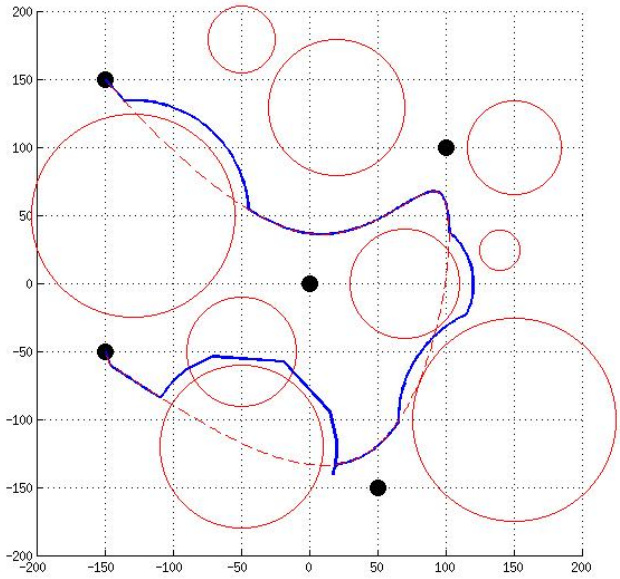

Figure 61: Smoothing Spline rho=1E7

As was stated before, having a large smoothing parameter in this environment is not helpful. Simply moving the splines outside the threat zones no longer works since the path may be pushed into another threat or a local minimum which would trap the robot. In this sort of environment the path of the splines are being distorted so much that they lose any benefit they have over Astar, therefore Astar would be the beneficial choice as the path generation method. It will always find the shortest path, it will not get stuck in local minimums, and will not get stuck in infinite loops.

4.5 Experimental Conclusion

Of the three methods studied for autonomous ground robotics, smoothing splines are the worst solution. In each of the tests the smoothing spline was never a more advantageous choice than both the B-Spline and Astar methods. There was a tendency for sharp turns, longer paths, and the fact that all previous points affect the current segment is also a negative. In real world situations sharp turns need to be avoided to fit system dynamics and realistically the beginning of a trip should stop affecting the current path at some point. It was shown that the Smoothing Spline is able to create a shorter path than the B-Spline method; however, this is only in simple circumstances and requires fine tuning for individual paths. These results hold true for real world situations because of the inability to prevent sharp turns and consistently longer path lengths. These issues would still be present in a full scale system.

B-Spline performed much better than the smoothing spline because they took in account the next three waypoints to intersect. This is shown in test 2 where the smoothing spline constantly overshot the waypoints where the B-Spline would swing out large enough to hit each point. However, this was not always true as in test 4 where there was a repeat of waypoints within the three references that B-Spline used. This caused a sharp overshoot in both the B-Spline and Smoothing Spline. B-Splines also held the advantage over Smoothing Splines when there are relatively few threats. If the smoothing parameter of Smoothing Splines is automated, its dynamic value will have an unknown effect on how strongly additional waypoints can change the path. This means the ability of an additional waypoint to redirect a path out of a threat zone is no longer guaranteed. With B-Splines guaranteed to intersect the waypoints, it is known that the new waypoint next to the threat will guide the spline around the threat. This is B-Splines strongest advantage in a hostile environment. In non-hostile environments the B-Spline was a shorter path than the smoothing spline. So in real world cases the B-Spline would be a better option than the smoothing spline. However it would depend on the environment whether B-Spline was better than Astar.

The area where both splines fail is in an environment with many threats. Neither method has the ability to take in additional information about areas to avoid and thus does not work as well as Astar in these situations. Any time the environment has more than just a sparse threat environment a search method similar to Astar will be much more reliable than a spline generation method. The downside to a search method is the much greater time needed for computation relative to the spline methods. In the end, the method used should be determined by the foreknowledge of the environment and the presence of obstacles or threats. This may allow for a method such as splines that can significantly reduce calculation times.

Figure 62: The Convoy

Figure 62 is showing three of the Khepera robots executing the convoy simulation. The khepera in the front is executing the leader controller and following the trajectories it generates while the back two kheperas are executing the follower controller. The white spheres placed on the top of the robots are markers used as position and orientation references for the Vicon system.

CHAPTER V

FUTURE WORK

Areas that can be addressed in future work could include additional trajectory methods or controllers. While three different trajectory methods were studied the obvious extension of this is to study additional methods and add a more sophisticated controller whose analysis and capabilities would be of great benefit.

In the current implementation the leader and followers are predefined. It would be beneficial to add the ability to dynamically organize connections allowing for the leader to be selected based on a given criteria.

A third area which could be expanded upon in future work is the ability for the Astar algorithm to assign a higher cost for turns that do not fit within the bounds of the robot. This would allow the algorithm to run at a higher resolution and remain within the robot dynamics.

REFERENCES

[1] X.C. Ding, A.R. Rahmani, and M. Egerstedt. Multi-uav convoy protection: an optimal approach to path planning and coordination. *Robotics, IEEE Transactions on*, 26(2):256–268, 2010.

[2] P. Twu, R. Chipalkatty, J.P. de la Croix, T. Ramachandran, J. Shively, M. Egerstedt, A. Rahmani, and R. Young. A hardware testbed for multi-uav collaborative ground convoy protection in dynamic environments. *AIAA Modeling and Simulation Technologies Conference*, 2011.

[3] R. Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114–121. IEEE, 1971.

[4] S. Koenig and M. Likhachev. Fast replanning for navigation in unknown terrain. *Robotics, IEEE Transactions on*, 21(3):354–363, 2005.

[5] J. Connors and G. Elkaim. Analysis of a spline based, obstacle avoiding path planning algorithm. In *Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th*, pages 2565–2569. IEEE, 2007.

[6] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, JP How, and G. Fiore. Real-time motion planning with applications to autonomous urban driving. *Control Systems Technology, IEEE Transactions on*, 17(5):1105–1118, 2009.

[7] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, et al. A perception-driven autonomous urban vehicle. *Journal of Field Robotics*, 25(10):727–774, 2008.

[8] L. Hunsberger and B.J. Grosz. A combinatorial auction for collaborative planning. In *MultiAgent Systems, 2000. Proceedings. Fourth International Conference on*, pages 151–158. IEEE, 2000.

[9] Magnus Egerstedt. *Control theoretic splines : optimal control, statistics, and path planning*. Princeton University Press, 2010.